ESP32

Technical Reference Manual

Version 3.5
Espressif Systems

www.espressif.com

About This Manual

The **ESP32 Technical Reference Manual** is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32 memory and peripherals.

For pin definition, electrical characteristics and package information, please see *ESP32 Datasheet*.

Revision History

For any changes to this document over time, please refer to the last page.

Related Resources

Additional documentation and other resources about ESP32 can be accessed here: ESP32 Resources.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice. THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to the use of information in this document, is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein. The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Contents

List of Tables

List of Figures

1. System and Memory

1.1 Introduction

The ESP32 is a dual-core system with two Harvard Architecture Xtensa LX6 CPUs. All embedded memory, external memory and peripherals are located on the data bus and/or the instruction bus of these CPUs.

With some minor exceptions (see below), the address mapping of two CPUs is symmetric, meaning that they use the same addresses to access the same memory. Multiple peripherals in the system can access embedded memory via DMA.

The two CPUs are named "PRO_CPU" and "APP_CPU" (for "protocol" and "application"), however, for most purposes the two CPUs are interchangeable.

1.2 Features

- Address Space
  - Symmetric address mapping
  - 4 GB (32-bit) address space for both data bus and instruction bus
  - 1296 KB embedded memory address space
  - 19704 KB external memory address space
  - 512 KB peripheral address space
  - Some embedded and external memory regions can be accessed by either data bus or instruction bus
  - 328 KB DMA address space
- Embedded Memory
  - 448 KB Internal ROM
  - 520 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- External Memory
  Off-chip SPI memory can be mapped into the available address space as external memory. Parts of the embedded memory can be used as transparent cache for this external memory.
  - Supports up to 16 MB off-Chip SPI Flash.
  - Supports up to 8 MB off-Chip SPI SRAM.
- Peripherals
  - 41 peripherals
- DMA
  - 13 modules are capable of DMA operation

The block diagram in Figure 1 illustrates the system structure, and the block diagram in Figure 2 illustrates the address map structure.

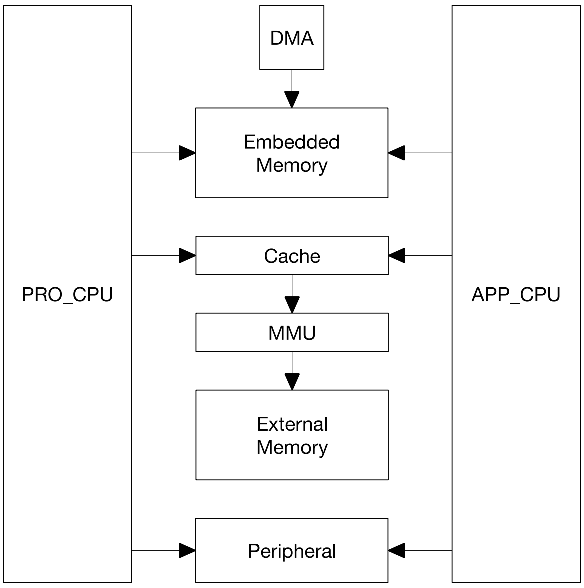


Figure 1: System Structure

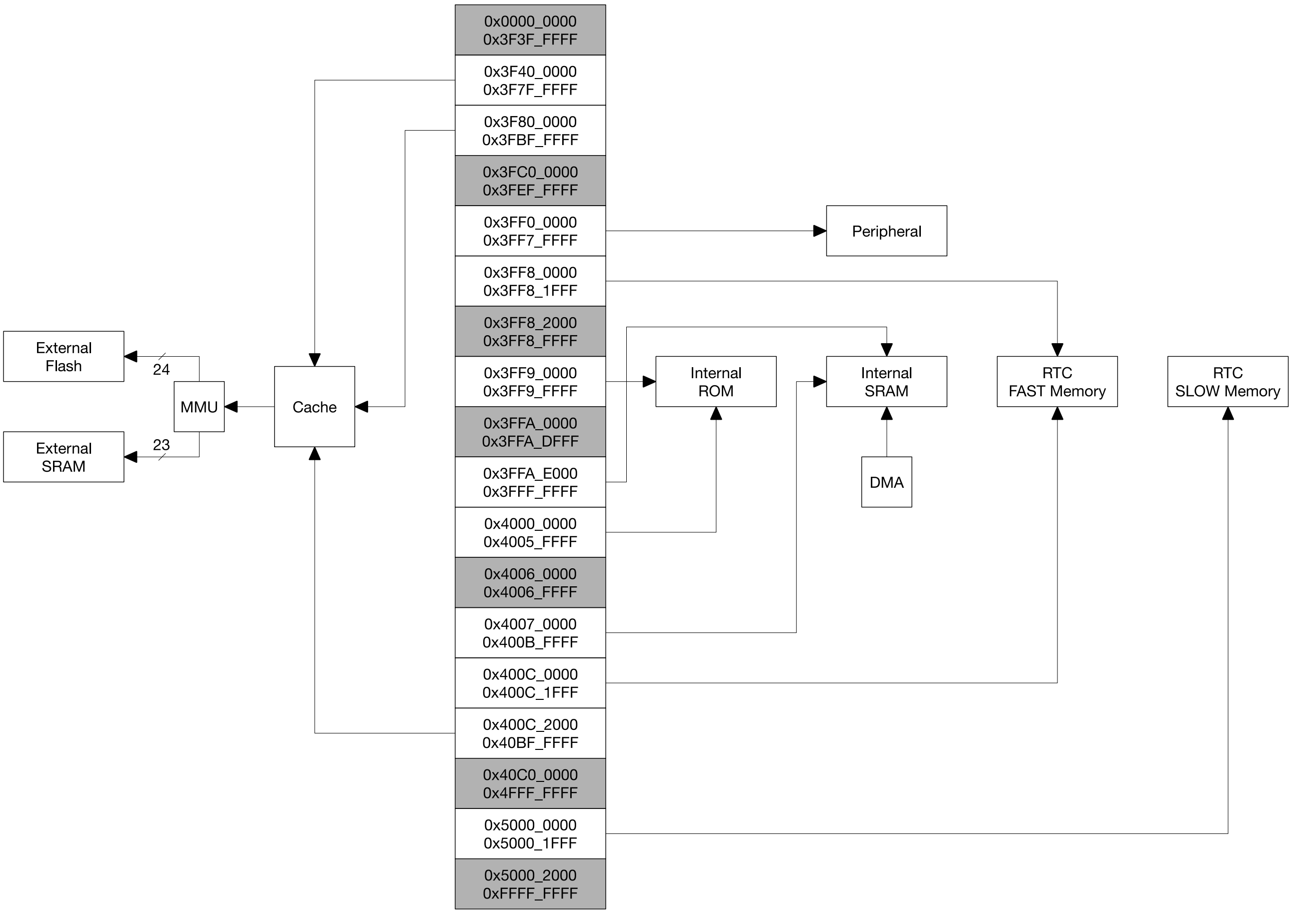


Figure 2: System Address Mapping

1.3 Functional Description

1.3.1 Address Mapping

Each of the two Harvard Architecture Xtensa LX6 CPUs has 4 GB (32-bit) address space. Address spaces are symmetric between the two CPUs.

Addresses below 0x4000_0000 are serviced using the data bus. Addresses in the range 0x4000_0000 ~ 0x4FFF_FFFF are serviced using the instruction bus. Finally, addresses over and including 0x5000_0000 are shared by the data and instruction bus.

The data bus and instruction bus are both little-endian: for example, byte addresses 0x0, 0x1, 0x2, 0x3 access the least significant, second least significant, second most significant, and the most significant bytes of the 32-bit word stored at the 0x0 address, respectively. The CPU can access data bus addresses via aligned or non-aligned byte, half-word and word read-and-write operations. The CPU can read and write data through the instruction bus, but only in a **word aligned manner**; non-word-aligned access will cause a CPU exception.

Each CPU can directly access embedded memory through both the data bus and the instruction bus, external memory which is mapped into the address space (via transparent caching & MMU), and peripherals. Table 1 illustrates address ranges that can be accessed by each CPU's data bus and instruction bus.

Some embedded memories and some external memories can be accessed via the data bus or the instruction bus. In these cases, the same memory is available to either of the CPUs at two address ranges.

Table 1: Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3F3F_FFFF		Reserved
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Memory
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External Memory
	0x3FC0_0000	0x3FEF_FFFF	3 MB	Reserved
Data	0x3FF0_0000	0x3FF7_FFFF	512 KB	Peripheral
Data	0x3FF8_0000	0x3FFF_FFFF	512 KB	Embedded Memory
Instruction	0x4000_0000	0x400C_1FFF	776 KB	Embedded Memory
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Memory
	0x40C0_0000	0x4FFF_FFFF	244 MB	Reserved
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	Embedded Memory
	0x5000_2000	0xFFFF_FFFF		Reserved

1.3.2 Embedded Memory

The Embedded Memory consists of four segments: internal ROM (448 KB), internal SRAM (520 KB), RTC FAST memory (8 KB) and RTC SLOW memory (8 KB).

The 448 KB internal ROM is divided into two parts: Internal ROM 0 (384 KB) and Internal ROM 1 (64 KB). The 520 KB internal SRAM is divided into three parts: Internal SRAM 0 (192 KB), Internal SRAM 1 (128 KB), and Internal SRAM 2 (200 KB). RTC FAST Memory and RTC SLOW Memory are both implemented as SRAM.

Table 2 lists all embedded memories and their address ranges on the data and instruction buses.

Table 2: Embedded Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

1.3.2.1 Internal ROM 0

The capacity of Internal ROM 0 is 384 KB. It is accessible by both CPUs through the address range 0x4000_0000 ~ 0x4005_FFFF, which is on the instruction bus.

The address range of the first 32 KB of the ROM 0 (0x4000_0000 ~ 0x4000_7FFF) can be remapped in order to access a part of Internal SRAM 1 that normally resides in a memory range of 0x400B_0000 ~ 0x400B_7FFF. While remapping, the 32 KB SRAM cannot be accessed by an address range of 0x400B_0000 ~ 0x400B_7FFF any more, but it can still be accessible through the data bus (0x3FFE_8000 ~ 0x3FFE_FFFF). This can be done on a per-CPU basis: setting bit 0 of register DPORT_PRO_BOOT_REMAP_CTRL_REG or DPORT_APP_BOOT_REMAP_CTRL_REG will remap SRAM for the PRO_CPU and APP_CPU, respectively.

1.3.2.2 Internal ROM 1

The capacity of Internal ROM 1 is 64 KB. It can be read by either CPU at an address range 0x3FF9_0000 ~ 0x3FF9_FFFF of the data bus.

1.3.2.3 Internal SRAM 0

The capacity of Internal SRAM 0 is 192 KB. Hardware can be configured to use the first 64 KB to cache external memory access. When not used as cache, the first 64 KB can be read and written by either CPU at addresses 0x4007_0000 ~ 0x4007_FFFF of the instruction bus. The remaining 128 KB can always be read and written by either CPU at addresses 0x4008_0000 ~ 0x4009_FFFF of instruction bus.

1.3.2.4 Internal SRAM 1

The capacity of Internal SRAM 1 is 128 KB. Either CPU can read and write this memory at addresses 0x3FFE_0000 ~ 0x3FFF_FFFF of the data bus, and also at addresses 0x400A_0000 ~ 0x400B_FFFF of the instruction bus.

The address range accessed via the instruction bus is in reverse order (word-wise) compared to access via the data bus. That is to say, address
0x3FFE_0000 and 0x400B_FFFC access the same word
0x3FFE_0004 and 0x400B_FFF8 access the same word
0x3FFE_0008 and 0x400B_FFF4 access the same word
……
0x3FFF_FFF4 and 0x400A_0008 access the same word
0x3FFF_FFF8 and 0x400A_0004 access the same word
0x3FFF_FFFC and 0x400A_0000 access the same word

The data bus and instruction bus of the CPU are still both little-endian, so the byte order of individual words is not reversed between address spaces. For example, address
0x3FFE_0000 accesses the least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0001 accesses the second least significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0002 accesses the second most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0003 accesses the most significant byte in the word accessed by 0x400B_FFFC.
0x3FFE_0004 accesses the least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0005 accesses the second least significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0006 accesses the second most significant byte in the word accessed by 0x400B_FFF8.
0x3FFE_0007 accesses the most significant byte in the word accessed by 0x400B_FFF8.
……
0x3FFF_FFF8 accesses the least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFF9 accesses the second least significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFA accesses the second most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFB accesses the most significant byte in the word accessed by 0x400A_0004.
0x3FFF_FFFC accesses the least significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFD accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFE accesses the second most significant byte in the word accessed by 0x400A_0000.
0x3FFF_FFFF accesses the most significant byte in the word accessed by 0x400A_0000.

Part of this memory can be remapped onto the ROM 0 address space. See Internal Rom 0 for more information.

1.3.2.5 Internal SRAM 2

The capacity of Internal SRAM 2 is 200 KB. It can be read and written by either CPU at addresses 0x3FFA_E000 ~ 0x3FFD_FFFF on the data bus.

1.3.2.6 DMA

DMA uses the same addressing as the CPU data bus to read and write Internal SRAM 1 and Internal SRAM 2. This means DMA uses an address range of 0x3FFE_0000 ~ 0x3FFF_FFFF to read and write Internal SRAM 1 and an address range of 0x3FFA_E000 ~ 0x3FFD_FFFF to read and write Internal SRAM 2.

In the ESP32, 13 peripherals are equipped with DMA. Table 3 lists these peripherals.

Table 3: Module with DMA

UART0	UART1	UART2
SPI1	SPI2	SPI3
I2S0	I2S1	
SDIO Slave	SDMMC	
EMAC		
BT	WIFI	

1.3.2.7 RTC FAST Memory

RTC FAST Memory is 8 KB of SRAM. It can be read and written by PRO_CPU only at an address range of 0x3FF8_0000 ~ 0x3FF8_1FFF on the data bus or at an address range of 0x400C_0000 ~ 0x400C_1FFF on the instruction bus. Unlike most other memory regions, RTC FAST memory cannot be accessed by the APP_CPU.

The two address ranges of PRO_CPU access RTC FAST Memory in the same order, so, for example, addresses 0x3FF8_0000 and 0x400C_0000 access the same word. **On the APP_CPU, these address ranges do not provide access to RTC FAST Memory or any other memory location.**

1.3.2.8 RTC SLOW Memory

RTC SLOW Memory is 8 KB of SRAM which can be read and written by either CPU at an address range of 0x5000_0000 ~ 0x5000_1FFF. This address range is shared by both the data bus and the instruction bus.

1.3.3 External Memory

The ESP32 can access external SPI flash and SPI SRAM as external memory. Table 4 provides a list of external memories that can be accessed by either CPU at a range of addresses on the data and instruction buses. When a CPU accesses external memory through the Cache and MMU, the cache will map the CPU's address to an external physical memory address (in the external memory's address space), according to the MMU settings. Due to this address mapping, the ESP32 can address up to 16 MB External Flash and 8 MB External SRAM.

Table 4: External Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3F40_0000	0x3F7F_FFFF	4 MB	External Flash	Read
Data	0x3F80_0000	0x3FBF_FFFF	4 MB	External SRAM	Read and Write
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x400C_2000	0x40BF_FFFF	11512 KB	External Flash	Read

1.3.4 Cache

As shown in Figure 3, each of the two CPUs in ESP32 has 32 KB of cache for accessing external storage. PRO CPU uses bit PRO_CACHE_ENABLE in register DPORT_PRO_CACHE_CTRL_REG to enable the Cache, while APP CPU uses bit APP_CACHE_ENABLE in register DPORT_APP_CACHE_CTRL_REG to enable the same function.

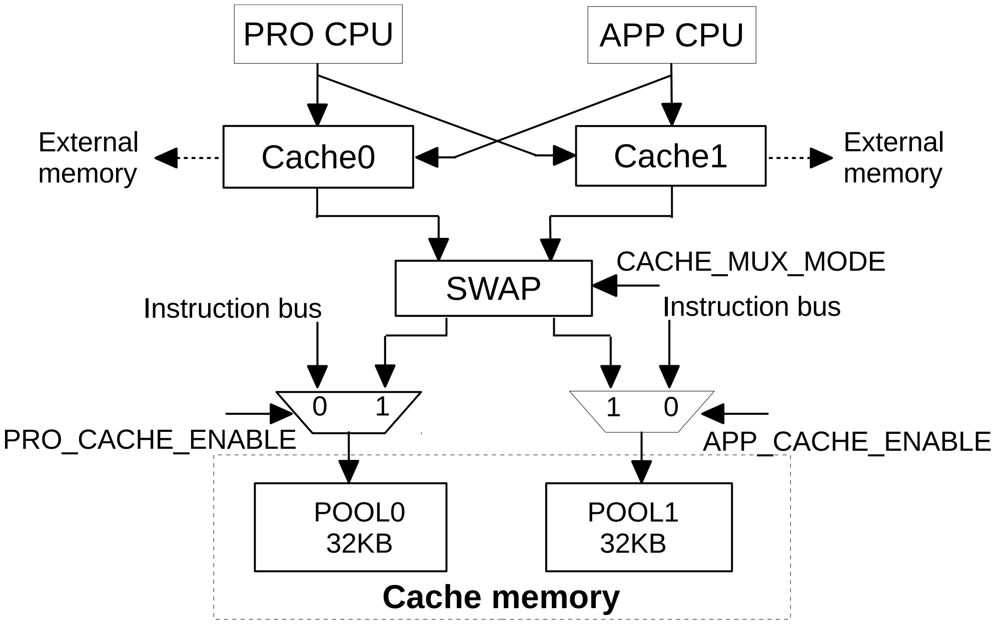


Figure 3: Cache Block Diagram

ESP32 uses a two-way set-associative cache. When the Cache function is to be used either by PRO CPU or APP CPU, bit CACHE_MUX_MODE[1:0] in register DPORT_CACHE_MUX_MODE_REG can be set to select POOL0 or POOL1 in the Internal SRAM0 as the cache memory. When both PRO CPU and APP CPU use the Cache function, POOL0 and POOL1 in the Internal SRAM0 will be used simultaneously as the cache memory, while they can also be used by the instruction bus. This is depicted in table 5 below.

Table 5: Cache memory mode

CACHE_MUX_MODE	POOL0	POOL1
0	PRO CPU	APP CPU
1	PRO CPU/APP CPU	-
2	-	PRO CPU/APP CPU
3	APP CPU	PRO CPU

As described in table 5, when bit CACHE_MUX_MODE is set to 1 or 2, PRO CPU and APP CPU cannot enable the Cache function at the same time. When the Cache function is enabled, POOL0 or POOL1 can only be used as the cache memory, and cannot be used by the instruction bus as well.

ESP32 Cache supports the Flush function. It is worth noting that when the Flush function is used, the data written in the cache will be disposed rather than being rewritten into the External SRAM. To enable the Flush function, first clear bit *x*_CACHE_FLUSH_ENA in register DPORT_*x*_CACHE_CTRL_REG, then set this bit to 1. Afterwards, the system hardware will set bit *x*_CACHE_FLUSH_DONE to 1, where *x* can be ”PRO” or ”APP”, indicating that the cache flush operation has been completed.

For more information about the address mapping of ESP32 Cache, please refer to Embedded Memory and External Memory.

1.3.5 Peripherals

The ESP32 has 41 peripherals. Table 6 specifically describes the peripherals and their respective address ranges. Nearly all peripheral modules can be accessed by either CPU at the same address with just a single exception; this being the PID Controller.

Table 6: Peripheral Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF0_0000	0x3FF0_0FFF	4 KB	DPort Register	
Data	0x3FF0_1000	0x3FF0_1FFF	4 KB	AES Accelerator	
Data	0x3FF0_2000	0x3FF0_2FFF	4 KB	RSA Accelerator	
Data	0x3FF0_3000	0x3FF0_3FFF	4 KB	SHA Accelerator	
Data	0x3FF0_4000	0x3FF0_4FFF	4 KB	Secure Boot	
	0x3FF0_5000	0x3FF0_FFFF	44 KB	Reserved	
Data	0x3FF1_0000	0x3FF1_3FFF	16 KB	Cache MMU Table	
	0x3FF1_4000	0x3FF1_EFFF	44 KB	Reserved	
Data	0x3FF1_F000	0x3FF1_FFFF	4 KB	PID Controller	Per-CPU peripheral
	0x3FF2_0000	0x3FF3_FFFF	128 KB	Reserved	
Data	0x3FF4_0000	0x3FF4_0FFF	4 KB	UART0	
	0x3FF4_1000	0x3FF4_1FFF	4 KB	Reserved	
Data	0x3FF4_2000	0x3FF4_2FFF	4 KB	SPI1	
Data	0x3FF4_3000	0x3FF4_3FFF	4 KB	SPI0	
Data	0x3FF4_4000	0x3FF4_4FFF	4 KB	GPIO	
	0x3FF4_5000	0x3FF4_7FFF	12 KB	Reserved	
Data	0x3FF4_8000	0x3FF4_8FFF	4 KB	RTC	
Data	0x3FF4_9000	0x3FF4_9FFF	4 KB	IO MUX	
	0x3FF4_A000	0x3FF4_AFFF	4 KB	Reserved	
Data	0x3FF4_B000	0x3FF4_BFFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF4_C000	0x3FF4_CFFF	4 KB	UDMA1	
	0x3FF4_D000	0x3FF4_EFFF	8 KB	Reserved	
Data	0x3FF4_F000	0x3FF4_FFFF	4 KB	I2S0	
Data	0x3FF5_0000	0x3FF5_0FFF	4 KB	UART1	
	0x3FF5_1000	0x3FF5_2FFF	8 KB	Reserved	

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF5_3000	0x3FF5_3FFF	4 KB	I2C0	
Data	0x3FF5_4000	0x3FF5_4FFF	4 KB	UDMA0	
Data	0x3FF5_5000	0x3FF5_5FFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF5_6000	0x3FF5_6FFF	4 KB	RMT	
Data	0x3FF5_7000	0x3FF5_7FFF	4 KB	PCNT	
Data	0x3FF5_8000	0x3FF5_8FFF	4 KB	SDIO Slave	One of three parts
Data	0x3FF5_9000	0x3FF5_9FFF	4 KB	LED PWM	
Data	0x3FF5_A000	0x3FF5_AFFF	4 KB	Efuse Controller	
Data	0x3FF5_B000	0x3FF5_BFFF	4 KB	Flash Encryption	
	0x3FF5_C000	0x3FF5_DFFF	8 KB	Reserved	
Data	0x3FF5_E000	0x3FF5_EFFF	4 KB	PWM0	
Data	0x3FF5_F000	0x3FF5_FFFF	4 KB	TIMG0	
Data	0x3FF6_0000	0x3FF6_0FFF	4 KB	TIMG1	
	0x3FF6_1000	0x3FF6_3FFF	12 KB	Reserved	
Data	0x3FF6_4000	0x3FF6_4FFF	4 KB	SPI2	
Data	0x3FF6_5000	0x3FF6_5FFF	4 KB	SPI3	
Data	0x3FF6_6000	0x3FF6_6FFF	4 KB	SYSCON	
Data	0x3FF6_7000	0x3FF6_7FFF	4 KB	I2C1	
Data	0x3FF6_8000	0x3FF6_8FFF	4 KB	SDMMC	
Data	0x3FF6_9000	0x3FF6_AFFF	8 KB	EMAC	
	0x3FF6_B000	0x3FF6_BFFF	4 KB	Reserved	
Data	0x3FF6_C000	0x3FF6_CFFF	4 KB	PWM1	
Data	0x3FF6_D000	0x3FF6_DFFF	4 KB	I2S1	
Data	0x3FF6_E000	0x3FF6_EFFF	4 KB	UART2	
Data	0x3FF6_F000	0x3FF6_FFFF	4 KB	PWM2	
Data	0x3FF7_0000	0x3FF7_0FFF	4 KB	PWM3	
	0x3FF7_1000	0x3FF7_4FFF	16 KB	Reserved	
Data	0x3FF7_5000	0x3FF7_5FFF	4 KB	RNG	
	0x3FF7_6000	0x3FF7_FFFF	40 KB	Reserved	

1.3.5.1 Asymmetric PID Controller Peripheral

There are two PID Controllers in the system. They serve the PRO_CPU and the APP_CPU, respectively. **The PRO_CPU and the APP_CPU can only access their own PID Controller and not that of their counterpart.** Each CPU uses the same memory range 0x3FF1_F000 ~ 3FF1_FFFF to access its own PID Controller.

1.3.5.2 Non-Contiguous Peripheral Memory Ranges

The SDIO Slave peripheral consists of three parts and the two CPUs use non-contiguous addresses to access these. The three parts are accessed at the address ranges 0x3FF4_B000 ~ 3FF4_BFFF, 0x3FF5_5000 ~ 3FF5_5FFF and 0x3FF5_8000 ~ 3FF5_8FFF of each CPU's data bus. Similarly to other peripherals, access to this peripheral is identical for both CPUs.

1.3.5.3 Memory Speed

The ROM as well as the SRAM are both clocked from CPU_CLK and can be accessed by the CPU in a single cycle. The RTC FAST memory is clocked from the APB_CLOCK and the RTC SLOW memory from the FAST_CLOCK, so access to these memories may be slower. DMA uses the APB_CLK to access memory.

Internally, the SRAM is organized in 32K-sized banks. Each CPU and DMA channel can simultaneously access the SRAM at full speed, provided they access addresses in different memory banks.

2. Interrupt Matrix

2.1 Introduction

The Interrupt Matrix embedded in the ESP32 independently allocates peripheral interrupt sources to the two CPUs' peripheral interrupts. This configuration is made to be highly flexible in order to meet many different needs.

2.2 Features

- Accepts 71 peripheral interrupt sources as input.
- Generates 26 peripheral interrupt sources per CPU as output (52 total).
- CPU NMI Interrupt Mask.
- Queries current interrupt status of peripheral interrupt sources.

The structure of the Interrupt Matrix is shown in Figure 4.

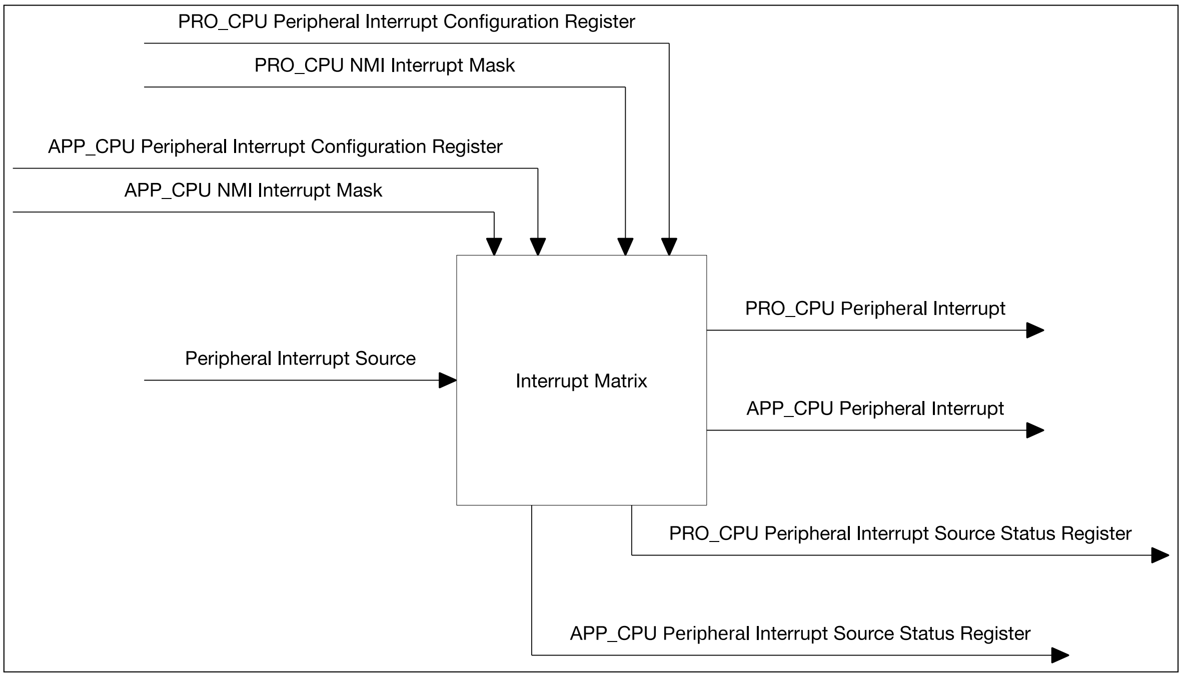

PRO_CPU Peripheral Interrupt Configuration Register
PRO_CPU NMI Interrupt Mask
APP_CPU Peripheral Interrupt Configuration Register
APP_CPU NMI Interrupt Mask
Peripheral Interrupt Source
Interrupt Matrix
PRO_CPU Peripheral Interrupt
APP_CPU Peripheral Interrupt
PRO_CPU Peripheral Interrupt Source Status Register
APP_CPU Peripheral Interrupt Source Status Register

Figure 4: Interrupt Matrix Structure

2.3 Functional Description

2.3.1 Peripheral Interrupt Source

ESP32 has 71 peripheral interrupt sources in total. All peripheral interrupt sources are listed in table 7. 67 of 71 ESP32 peripheral interrupt sources can be allocated to either CPU.

The four remaining peripheral interrupt sources are CPU-specific, two per CPU. GPIO_INTERRUPT_PRO and GPIO_INTERRUPT_PRO_NMI can only be allocated to PRO_CPU. GPIO_INTERRUPT_APP and GPIO_INTERRUPT_APP_NMI can only be allocated to APP_CPU. As a result, PRO_CPU and APP_CPU each have 69 peripheral interrupt sources.

Table 7: PRO_CPU, APP_CPU Interrupt Configuration

PRO_CPU						APP_CPU
Peripheral Interrupt Configuration Register			Peripheral Interrupt Source			
	Bit	Status Register Name	No.	Name	No.	Status Register Name
PRO_MAC_INTR_MAP_REG	0	PRO_INTR_STATUS_REG_0	0	MAC_INTR	0	APP_INTR_STATUS_REG_0
PRO_MAC_NMI_MAP_REG	1		1	MAC_NMI	1	
PRO_BB_INT_MAP_REG	2		2	BB_INT	2	
PRO_BT_MAC_INT_MAP_REG	3		3	BT_MAC_INT	3	
PRO_BT_BB_INT_MAP_REG	4		4	BT_BB_INT	4	
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5	
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6	
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5	
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6	
PRO_RWBLE_IRQ_MAP_REG	7		7	RWBLE_IRQ	7	
PRO_RWBT_NMI_MAP_REG	8		8	RWBT_NMI	8	
PRO_RWBLE_NMI_MAP_REG	9		9	RWBLE_NMI	9	
PRO_SLC0_INTR_MAP_REG	10		10	SLC0_INTR	10	
PRO_SLC1_INTR_MAP_REG	11		11	SLC1_INTR	11	
PRO_UHCI0_INTR_MAP_REG	12		12	UHCI0_INTR	12	
PRO_UHCI1_INTR_MAP_REG	13		13	UHCI1_INTR	13	
PRO_TG_T0_LEVEL_INT_MAP_REG	14		14	TG_T0_LEVEL_INT	14	
PRO_TG_T1_LEVEL_INT_MAP_REG	15		15	TG_T1_LEVEL_INT	15	
PRO_TG_WDT_LEVEL_INT_MAP_REG	16		16	TG_WDT_LEVEL_INT	16	
PRO_TG_LACT_LEVEL_INT_MAP_REG	17		17	TG_LACT_LEVEL_INT	17	
PRO_TG1_T0_LEVEL_INT_MAP_REG	18		18	TG1_T0_LEVEL_INT	18	
PRO_TG1_T1_LEVEL_INT_MAP_REG	19		19	TG1_T1_LEVEL_INT	19	
PRO_TG1_WDT_LEVEL_INT_MAP_REG	20		20	TG1_WDT_LEVEL_INT	20	
PRO_TG1_LACT_LEVEL_INT_MAP_REG	21		21	TG1_LACT_LEVEL_INT	21	
PRO_GPIO_INTERRUPT_PRO_MAP_REG	22		22	**GPIO_INTERRUPT_PRO** **GPIO_INTERRUPT_APP**	22	
PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	23		23	**GPIO_INTERRUPT_PRO_NMI** **GPIO_INTERRUPT_APP_NMI**	23	
PRO_CPU_INTR_FROM_CPU_0_MAP_REG	24		24	CPU_INTR_FROM_CPU_0	24	
PRO_CPU_INTR_FROM_CPU_1_MAP_REG	25		25	CPU_INTR_FROM_CPU_1	25	
PRO_CPU_INTR_FROM_CPU_2_MAP_REG	26		26	CPU_INTR_FROM_CPU_2	26	
PRO_CPU_INTR_FROM_CPU_3_MAP_REG	27		27	CPU_INTR_FROM_CPU_3	27	
PRO_SPI_INTR_0_MAP_REG	28		28	SPI_INTR_0	28	
PRO_SPI_INTR_1_MAP_REG	29		29	SPI_INTR_1	29	
PRO_SPI_INTR_2_MAP_REG	30		30	SPI_INTR_2	30	
PRO_SPI_INTR_3_MAP_REG	31		31	SPI_INTR_3	31	
PRO_I2S0_INT_MAP_REG	0	PRO_INTR_STATUS_REG_1	32	I2S0_INT	32	APP_INTR_STATUS_REG_1
PRO_I2S1_INT_MAP_REG	1		33	I2S1_INT	33	
PRO_UART_INTR_MAP_REG	2		34	UART_INTR	34	
PRO_UART1_INTR_MAP_REG	3		35	UART1_INTR	35	
PRO_UART2_INTR_MAP_REG	4		36	UART2_INTR	36	
PRO_SDIO_HOST_INTERRUPT_MAP_REG	5		37	SDIO_HOST_INTERRUPT	37	
PRO_EMAC_INT_MAP_REG	6		38	EMAC_INT	38	
PRO_PWM0_INTR_MAP_REG	7		39	PWM0_INTR	39	
PRO_PWM1_INTR_MAP_REG	8		40	PWM1_INTR	40	
PRO_PWM2_INTR_MAP_REG	9		41	PWM2_INTR	41	
PRO_PWM3_INTR_MAP_REG	10		42	PWM3_INTR	42	
PRO_LEDC_INT_MAP_REG	11		43	LEDC_INT	43	
PRO_EFUSE_INT_MAP_REG	12		44	EFUSE_INT	44	
PRO_CAN_INT_MAP_REG	13		45	CAN_INT	45	
PRO_RTC_CORE_INTR_MAP_REG	14		46	RTC_CORE_INTR	46	
PRO_RMT_INTR_MAP_REG	15		47	RMT_INTR	47	
PRO_PCNT_INTR_MAP_REG	16		48	PCNT_INTR	48	
PRO_I2C_EXT0_INTR_MAP_REG	17		49	I2C_EXT0_INTR	49	
PRO_I2C_EXT1_INTR_MAP_REG	18		50	I2C_EXT1_INTR	50	
PRO_RSA_INTR_MAP_REG	19		51	RSA_INTR	51	
PRO_SPI1_DMA_INT_MAP_REG	20		52	SPI1_DMA_INT	52	

PRO_CPU						APP_CPU
Peripheral Interrupt Configuration Register	Peripheral Interrupt Source					
	Status Register		No.	Name	No.	Status Register Name
	Bit	Name				
PRO_SPI2_DMA_INT_MAP_REG	21	PRO_INTR_STATUS_REG_1	53	SPI2_DMA_INT	53	APP_INTR_STATUS_REG_1
PRO_SPI3_DMA_INT_MAP_REG	22		54	SPI3_DMA_INT	54	
PRO_WDG_INT_MAP_REG	23		55	WDG_INT	55	
PRO_TIMER_INT1_MAP_REG	24		56	TIMER_INT1	56	
PRO_TIMER_INT2_MAP_REG	25		57	TIMER_INT2	57	
PRO_TG_T0_EDGE_INT_MAP_REG	26		58	TG_T0_EDGE_INT	58	
PRO_TG_T1_EDGE_INT_MAP_REG	27		59	TG_T1_EDGE_INT	59	
PRO_TG_WDT_EDGE_INT_MAP_REG	28		60	TG_WDT_EDGE_INT	60	
PRO_TG_LACT_EDGE_INT_MAP_REG	29		61	TG_LACT_EDGE_INT	61	
PRO_TG1_T0_EDGE_INT_MAP_REG	30		62	TG1_T0_EDGE_INT	62	
PRO_TG1_T1_EDGE_INT_MAP_REG	31		63	TG1_T1_EDGE_INT	63	
PRO_TG1_WDT_EDGE_INT_MAP_REG	0	PRO_INTR_STATUS_REG_2	64	TG1_WDT_EDGE_INT	64	APP_INTR_STATUS_REG_2
PRO_TG1_LACT_EDGE_INT_MAP_REG	1		65	TG1_LACT_EDGE_INT	65	
PRO_MMU_IA_INT_MAP_REG	2		66	MMU_IA_INT	66	
PRO_MPU_IA_INT_MAP_REG	3		67	MPU_IA_INT	67	
PRO_CACHE_IA_INT_MAP_REG	4		68	CACHE_IA_INT	68	

2.3.2 CPU Interrupt

Both of the two CPUs (PRO and APP) have 32 interrupts each, of which 26 are peripheral interrupts. All interrupts in a CPU are listed in Table 8.

Table 8: CPU Interrupts

No.	Category	Type	Priority Level
0	Peripheral	Level-Triggered	1
1	Peripheral	Level-Triggered	1
2	Peripheral	Level-Triggered	1
3	Peripheral	Level-Triggered	1
4	Peripheral	Level-Triggered	1
5	Peripheral	Level-Triggered	1
6	Internal	Timer.0	1
7	Internal	Software	1
8	Peripheral	Level-Triggered	1
9	Peripheral	Level-Triggered	1
10	Peripheral	Edge-Triggered	1
11	Internal	Profiling	3
12	Peripheral	Level-Triggered	1
13	Peripheral	Level-Triggered	1
14	Peripheral	NMI	NMI
15	Internal	Timer.1	3
16	Internal	Timer.2	5
17	Peripheral	Level-Triggered	1
18	Peripheral	Level-Triggered	1
19	Peripheral	Level-Triggered	2
20	Peripheral	Level-Triggered	2
21	Peripheral	Level-Triggered	2
22	Peripheral	Edge-Triggered	3
23	Peripheral	Level-Triggered	3
24	Peripheral	Level-Triggered	4
25	Peripheral	Level-Triggered	4
26	Peripheral	Level-Triggered	5
27	Peripheral	Level-Triggered	3
28	Peripheral	Edge-Triggered	4
29	Internal	Software	3
30	Peripheral	Edge-Triggered	4
31	Peripheral	Level-Triggered	5

2.3.3 Allocate Peripheral Interrupt Sources to Peripheral Interrupt on CPU

In this section:

- Source_X stands for any particular peripheral interrupt source.
- PRO_X_MAP_REG (or APP_X_MAP_REG) stands for any particular peripheral interrupt configuration

register of the PRO_CPU (or APP_CPU). The peripheral interrupt configuration register corresponds to the peripheral interrupt source Source_X. In Table 7 the registers listed under “PRO_CPU (APP_CPU) - Peripheral Interrupt Configuration Register” correspond to the peripheral interrupt sources listed in “Peripheral Interrupt Source - Name”.

- Interrupt_P stands for CPU peripheral interrupt, numbered as Num_P. Num_P can take the ranges 0 ~ 5, 8 ~ 10, 12 ~ 14, 17 ~ 28, 30 ~ 31.
- Interrupt_I stands for the CPU internal interrupt numbered as Num_I. Num_I can take values 6, 7, 11, 15, 16, 29.

Using this terminology, the possible operations of the Interrupt Matrix controller can be described as follows:

- **Allocate peripheral interrupt source Source_X to CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG or APP_X_MAP_REG to Num_P. Num_P can be any CPU peripheral interrupt number. CPU interrupts can be shared between multiple peripherals (see below).
- **Disable peripheral interrupt source Source_X for CPU (PRO_CPU or APP_CPU)**
  Set PRO_X_MAP_REG or APP_X _MAP_REG for peripheral interrupt source to any Num_I. The specific choice of internal interrupt number does not change behaviour, as none of the interrupt numbered as Num_I is connected to either CPU.
- **Allocate multiple peripheral sources Source_X*n* ORed to PRO_CPU (APP_CPU) peripheral interrupt**
  Set multiple PRO_X*n*_MAP_REG (APP_X*n*_MAP_REG) to the same Num_P. Any of these peripheral interrupts will trigger CPU Interrupt_P.

2.3.4 CPU NMI Interrupt Mask

The Interrupt Matrix temporarily masks all peripheral interrupt sources allocated to PRO_CPU’s (or APP_CPU’s) NMI interrupt, if it receives the signal PRO_CPU NMI Interrupt Mask (or APP_CPU NMI Interrupt Mask) from the peripheral PID Controller, respectively.

2.3.5 Query Current Interrupt Status of Peripheral Interrupt Source

The current interrupt status of a peripheral interrupt source can be read via the bit value in PRO_INTR_STATUS_REG_*n* (APP_INTR_STATUS_REG_*n*), as shown in the mapping in Table 7.

3. Reset and Clock

3.1 System Reset

3.1.1 Introduction

The ESP32 has three reset levels: CPU reset, Core reset, and System reset. None of these reset levels clear the RAM. Figure 5 shows the subsystems included in each reset level.

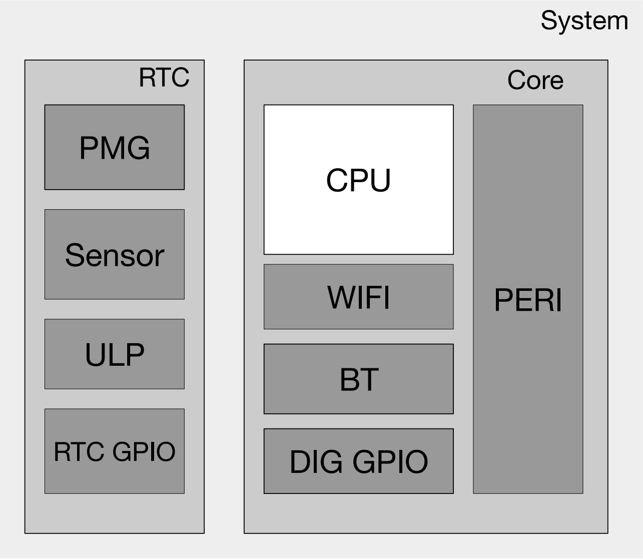


Figure 5: System Reset

- CPU reset: Only resets the registers of one or both of the CPU cores.
- Core reset: Resets all the digital registers, including CPU cores, external GPIO and digital GPIO. The RTC is not reset.
- System reset: Resets all the registers on the chip, including those of the RTC.

3.1.2 Reset Source

While most of the time the APP_CPU and PRO_CPU will be reset simultaneously, some reset sources are able to reset only one of the two cores. The reset reason for each core can be looked up individually: the PRO_CPU reset reason will be stored in RTC_CNTL_RESET_CAUSE_PROCPU, the reset reason for the APP_CPU in RTC_CNTL_RESET_CAUSE_APPCPU. Table 9 shows the possible reset reason values that can be read from these registers.

Table 9: PRO_CPU and APP_CPU Reset Reason Values

PRO	APP	Source	Reset Type	Note
0x01	0x01	Chip Power On Reset	System Reset	-
0x10	0x10	RWDT System Reset	System Reset	See WDT Chapter.
0x0F	0x0F	Brown Out Reset	System Reset	See Power Management Chapter.
0x03	0x03	Software System Reset	Core Reset	Configure RTC_CNTL_SW_SYS_RST register.
0x05	0x05	Deep Sleep Reset	Core Reset	See Power Management Chapter.
0x07	0x07	MWDT0 Global Reset	Core Reset	See WDT Chapter.

PRO	APP	APP Source	Reset Type	Note
0x08	0x08	MWDT1 Global Reset	Core Reset	See WDT Chapter.
0x09	0x09	RWDT Core Reset	Core Reset	See WDT Chapter.
0x0B	-	MWDT0 CPU Reset	CPU Reset	See WDT Chapter.
0x0C	-	Software CPU Reset	CPU Reset	Configure RTC_CNTL_SW_APPCPU_RST register.
-	0x0B	MWDT1 CPU Reset	CPU Reset	See WDT Chapter.
-	0x0C	Software CPU Reset	CPU Reset	Configure RTC_CNTL_SW_APPCPU_RST register.
0x0D	0x0D	RWDT CPU Reset	CPU Reset	See WDT Chapter.
-	0xE	PRO CPU Reset	CPU Reset	Indicates that the PRO CPU has independently reset the APP CPU by configuring the DPORT_APPCPU_RESETTING register.

3.2 System Clock

3.2.1 Introduction

The ESP32 integrates multiple clock sources for the CPU cores, the peripherals and the RTC. These clocks can be configured to meet different requirements. Figure 6 shows the system clock structure.

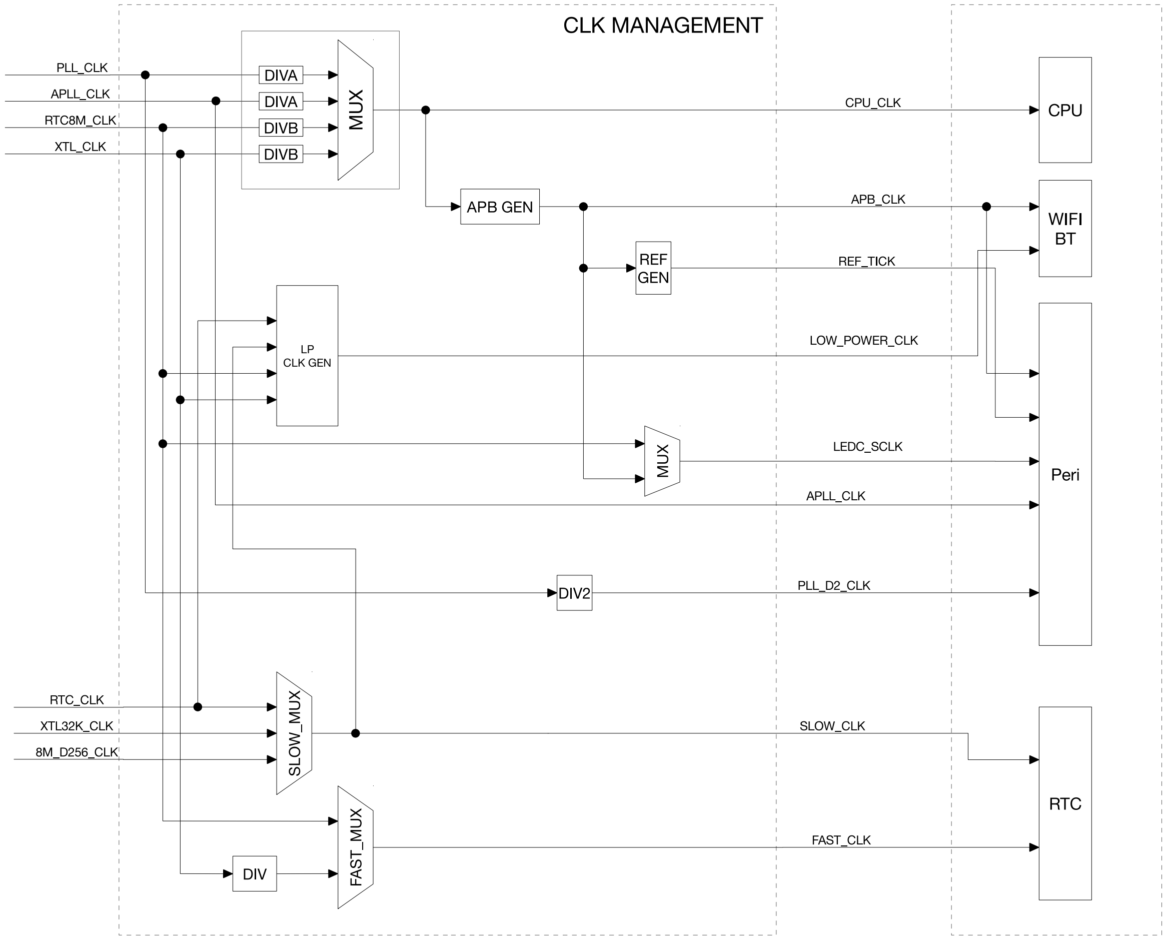


Figure 6: System Clock

3.2.2 Clock Source

The ESP32 can use an external crystal oscillator, an internal PLL or an oscillating circuit as a clock source. Specifically, the clock sources available are:

- High Speed Clocks
  - PLL_CLK is an internal PLL clock with a frequency of 320 MHz.
  - XTL_CLK is a clock signal generated using an external crystal with a frequency range of 2 ~ 40 MHz.
- Low Power Clocks
  - XTL32K_CLK is a clock generated using an external crystal with a frequency of 32 KHz.
  - RTC8M_CLK is an internal clock with a default frequency of 8 MHz. This frequency is adjustable.
  - RTC8M_D256_CLK is divided from RTC8M_CLK 256. Its frequency is (RTC8M_CLK / 256). With the default RTC8M_CLK frequency of 8 MHz, this clock runs at 31.250 KHz.
  - RTC_CLK is an internal low power clock with a default frequency of 150 KHz. This frequency is adjustable.
- Audio Clock
  - APLL_CLK is an internal Audio PLL clock with a frequency range of 16 ~ 128 MHz.

3.2.3 CPU Clock

As Figure 6 shows, CPU_CLK is the master clock for both CPU cores. CPU_CLK clock can be as high as 160 MHz when the CPU is in high performance mode. Alternatively, the CPU can run at lower frequencies to reduce power consumption.

The CPU_CLK clock source is determined by the RTC_CNTL_SOC_CLK_SEL register. PLL_CLK, APLL_CLK, RTC8M_CLK and XTL_CLK can be set as the CPU_CLK source; see Table 10 and 11.

Table 10: CPU_CLK Source

RTC_CNTL_SOC_CLK_SEL Value	Clock Source
0	XTL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

Table 11: CPU_CLK Derivation

Clock Source	SEL*	CPU Clock
0 / XTL_CLK	-	CPU_CLK = XTL_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTRL_PRE_DIV_CNT range is 0 ~ 1023. Default is 0.
1 / PLL_CLK	0	CPU_CLK = PLL_CLK / 4 CPU_CLK frequency is 80 MHz
1 / PLL_CLK	1	CPU_CLK = PLL_CLK / 2 CPU_CLK frequency is 160 MHz
2 / RTC8M_CLK	-	CPU_CLK = RTC8M_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTRL_PRE_DIV_CNT range is 0 ~ 1023. Default is 0.
3 / APLL_CLK	0	CPU_CLK = APLL_CLK / 4
3 / APLL_CLK	1	CPU_CLK = APLL_CLK / 2

*SEL: DPORT_CPUPERIOD _SEL value

3.2.4 Peripheral Clock

Peripheral clocks include APB_CLK, REF_TICK, LEDC_SCLK, APLL_CLK and PLL_D2_CLK.
Table 12 shows which clocks can be used by which peripherals.

Table 12: Peripheral Clock Usage

Peripherals	APB_CLK	REF_TICK	LEDC_SCLK	APLL_CLK	PLL_D2_CLK
EMAC	Y	N	N	Y	N
TIMG	Y	N	N	N	N
I2S	Y	N	N	Y	Y
UART	Y	Y	N	N	N
RMT	Y	Y	N	N	N
LED PWM	Y	Y	Y	N	N
PWM	Y	N	N	N	N
I2C	Y	N	N	N	N
SPI	Y	N	N	N	N
PCNT	Y	N	N	N	N
Efuse Controller	Y	N	N	N	N
SDIO Slave	Y	N	N	N	N
SDMMC	Y	N	N	N	N

3.2.4.1 APB_CLK Source

The APB_CLK is derived from CPU_CLK as detailed in Table 13. The division factor depends on the CPU_CLK source.

Table 13: APB_CLK Derivation

CPU_CLK Source	APB_CLK
PLL_CLK	PLL_CLK / 4
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

3.2.4.2 REF_TICK Source

REF_TICK is derived from APB_CLK via a divider. The divider value used depends on the APB_CLK source, which in turn depends on the CPU_CLK source.

By configuring correct divider values for each APB_CLK source, the user can ensure that the REF_TICK frequency does not change when CPU_CLK changes source, causing the APB_CLK frequency to change.

Clock divider registers are shown in Table 14.

Table 14: REF_TICK Derivation

CPU_CLK & APB_CLK Source	Clock Divider Register
PLL_CLK	APB_CTRL_PLL_TICK_NUM
XTAL_CLK	APB_CTRL_XTAL_TICK_NUM
APLL_CLK	APB_CTRL_APLL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

3.2.4.3 LEDC_SCLK Source

The LEDC_SCLK clock source is selected by the LEDC_APB_CLK_SEL register, as shown in Table 15.

Table 15: LEDC_SCLK Derivation

LEDC_APB_CLK_SEL Value	LEDC_SCLK Source
0	RTC8M_CLK
1	APB_CLK

3.2.4.4 APLL_SCLK Source

The APLL_CLK is sourced from PLL_CLK, with its output frequency configured using the APLL configuration registers.

3.2.4.5 PLL_D2_CLK Source

PLL_D2_CLK is half the PLL_CLK frequency.

3.2.4.6 Clock Source Considerations

Most peripherals will operate using the APB_CLK frequency as a reference. When this frequency changes, the peripherals will need to update their clock configuration to operate at the same frequency after the change. Peripherals accessing REF_TICK can continue operating normally when switching clock sources, without changing clock source. Please see Table 12 for details.

The LED PWM module can use RTC8M_CLK as a clock source when APB_CLK is disabled. In other words, when the system is in low-power consumption mode (see Power Management Chapter), normal peripherals will be halted (APB_CLK is turned off), but the LED PWM can work normally via RTC8M_CLK.

3.2.5 Wi-Fi BT Clock

Wi-Fi and BT can only operate if APB_CLK uses PLL_CLK as its clock source. Suspending PLL_CLK requires Wi-Fi and BT to both have entered low-power consumption mode first.

For LOW_POWER_CLK, one of RTC_CLK, SLOW_CLK, RTC8M_CLK or XTL_CLK can be selected as the low-power consumption mode clock source for Wi-Fi and BT.

3.2.6 RTC Clock

The clock sources of SLOW_CLK and FAST_CLK are low-frequency clocks. The RTC module can operate when most other clocks are stopped.

SLOW_CLK is used to clock the Power Management module. It can be sourced from RTC_CLK, XTL32K_CLK or RTC8M_D256_CLK

FAST_CLK is used to clock the On-chip Sensor module. It can be sourced from a divided XTL_CLK or from RTC8M_CLK.

3.2.7 Audio PLL

The operation of audio and other time-critical data-transfer applications requires highly-configurable, low-jitter, and accurate clock sources. The clock sources derived from system clocks that serve digital peripherals may carry jitter and, therefore, they do not support a high-precision clock frequency setting.

Providing an integrated precision clock source can minimize system cost. To this end, ESP32 integrates an audio PLL intended for I2S peripherals. More details on how to clock the I2S module, using an APLL clock, can be found in Chapter I2S. The Audio PLL formula is as follows:

$$f_{\text{out}} = \frac{f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4)}{2(odiv + 2)}$$

The parameters of this formula are defined below:

- f_{xtal}: the frequency of the crystal oscillator, usually 40 MHz;
- sdm0: the value is 0 ~ 255;
- sdm1: the value is 0 ~ 255;
- sdm2: the value is 0 ~ 63;
- odir: the value is 0 ~ 31;

The operating frequency range of the numerator is 350 MHz ~ 500 MHz:

$$350MHz < f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4) < 500MHz$$

Please note that sdm1 and sdm0 are not available on revision0 of ESP32. Please consult the silicon revision in ECO and Workarounds for Bugs in ESP32 for further details.

Audio PLL can be manually enabled or disabled via registers RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD, respectively. Disabling it takes priority over enabling it. When RTC_CNTL_PLLA_FORCE_PU and RTC_CNTL_PLLA_FORCE_PD are 0, PLL will follow the state of the system, i.e., when the system enters sleep mode, PLL will be disabled automatically; when the system wakes up, PLL will be enabled automatically.

4. IO_MUX and GPIO Matrix

4.1 Overview

The ESP32 chip features 34 physical GPIO pads. Each pad can be used as a general-purpose I/O, or be connected to an internal peripheral signal. The IO_MUX, RTC IO_MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pads. Together these systems provide highly configurable I/O.

Note that the I/O GPIO pads are 0-19, 21-23, 25-27, 32-39, while the output GPIOs are 0-19, 21-23, 25-27, 32-33. GPIO pads 34-39 are input-only.

This chapter describes the signal selection and connection between the digital pads (FUNC_SEL, IE, OE, WPU, WDU, etc.), 162 peripheral input and 176 output signals (control signals: SIG_IN_SEL, SIG_OUT_SEL, IE, OE, etc.), fast peripheral input/output signals (control signals: IE, OE, etc.), and RTC IO_MUX.

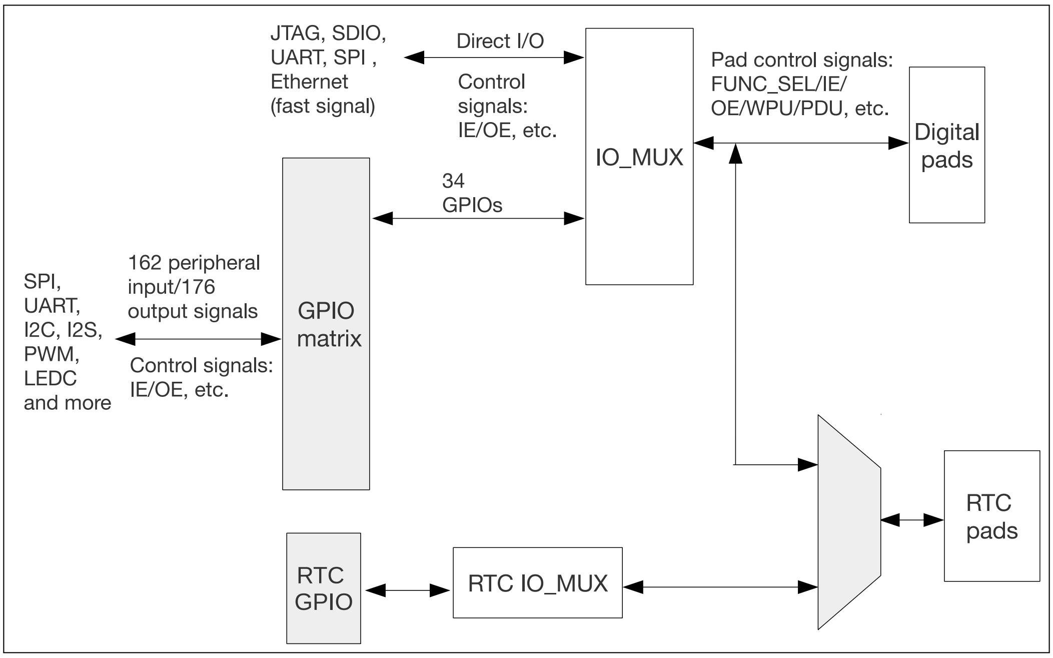


Figure 7: IO_MUX, RTC IO_MUX and GPIO Matrix Overview

1. The IO_MUX contains one register per GPIO pad. Each pad can be configured to perform a ”GPIO” function (when connected to the GPIO Matrix) or a direct function (bypassing the GPIO Matrix). Some high-speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO_MUX is used to connect these pads directly to the peripheral.)

   See Section 4.10 for a list of IO_MUX functions for each I/O pad.

2. The GPIO Matrix is a full-switching matrix between the peripheral input/output signals and the pads.

   - For input to the chip: Each of the 162 internal peripheral inputs can select any GPIO pad as the input source.

   - For output from the chip: The output signal of each of the 34 GPIO pads can be from one of the 176 peripheral output signals.

   See Section 4.9 for a list of GPIO Matrix peripheral signals.

3. RTC IO_MUX is used to connect GPIO pads to their low-power and analog functions. Only a subset of GPIO pads have these optional ”RTC” functions.

See Section 4.11 for a list of RTC IO_MUX functions.

4.2 Peripheral Input via GPIO Matrix

4.2.1 Summary

To receive a peripheral input signal via the GPIO Matrix, the GPIO Matrix is configured to source the peripheral signal's input index (0-18, 23-36, 39-58, 61-90, 95-124, 140-155, 164-181, 190-195, 198-206) from one of the 34 GPIOs (0-19, 21-23, 25-27, 32-39).

The input signal is read from the GPIO pad through the IO_MUX. The IO_MUX must be configured to set the chosen pad to "GPIO" function. This causes the GPIO pad input signal to be routed into the GPIO Matrix, which in turn routes it to the selected peripheral input.

4.2.2 Functional Description

Figure 8 shows the logic for input selection via GPIO Matrix.

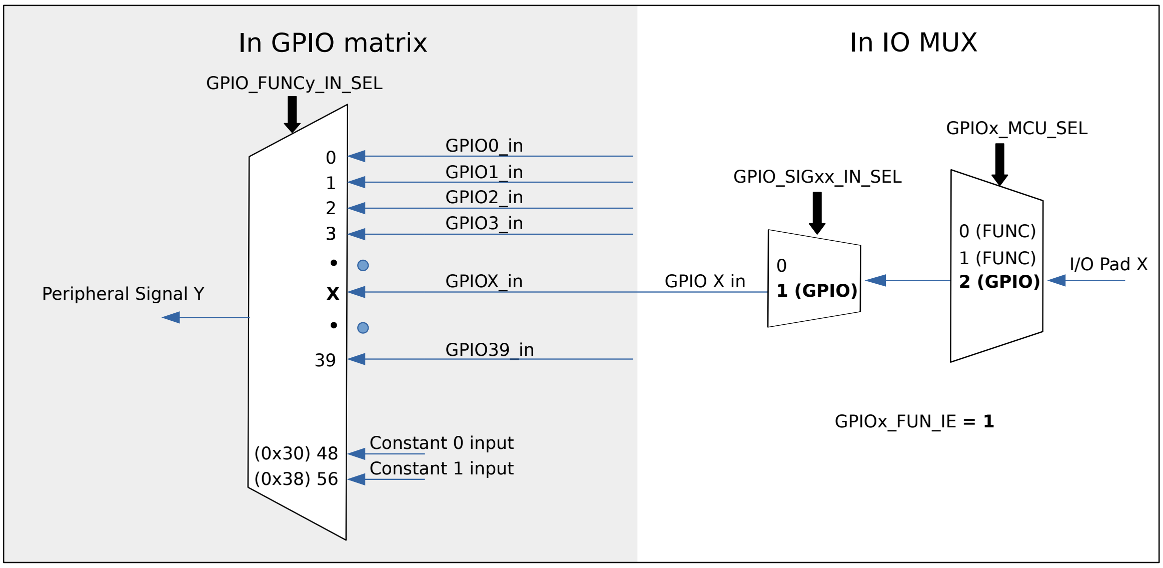


Figure 8: Peripheral Input via IO_MUX, GPIO Matrix

To read GPIO pad *X* into peripheral signal *Y*, follow the steps below:

1. Configure the GPIO_FUNC*y*_IN_SEL_CFG register corresponding to peripheral signal *Y* in the GPIO Matrix:
   - Set the GPIO_FUNC*x*_IN_SEL field in this register, corresponding to the GPIO pad *X* to read from. Clear all other fields corresponding to other GPIO pads.
2. Configure the GPIO_FUNC*x*_OUT_SEL_CFG register and clear the GPIO_ENABLE_DATA[*x*] field corresponding to GPIO pad *X* in the GPIO Matrix:
   - Set the GPIO_FUNC*x*_OEN_SEL bit in the GPIO_FUNC*x*_OUT_SEL_CFG register to force the pin's output state to be determined always by the GPIO_ENABLE_DATA[*x*] field.
   - The GPIO_ENABLE_DATA[*x*] field is a bit in either GPIO_ENABLE_REG (GPIOs 0-31) or GPIO_ENABLE1_REG (GPIOs 32-39). Clear this bit to disable the output driver for the GPIO pad.
3. Configure the IO_MUX to select the GPIO Matrix. Set the IO_MUX_*x*_REG register corresponding to GPIO pad *X* as follows:

  - Set the function field (IO_*x*_MCU_SEL) to the IO_MUX function corresponding to GPIO *X* (this is Function #3—numeric value 2—for all pins).
  - Enable the input by setting the FUN_IE bit.
  - Set or clear the FUN_WPU and FUN_WPD bits, as desired, to enable/disable internal pull-up/pull-down resistors.

Notes:

- One input pad can be connected to multiple input_signals.
- The input signal can be inverted with GPIO_FUNC*x*_IN_INV_SEL.
- It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pad. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
  - When GPIO_FUNC*x*_IN_SEL is 0x30, input_signal_*x* is always 0.
  - When GPIO_FUNC*x*_IN_SEL is 0x38, input_signal_*x* is always 1.

For example, to connect RMT peripheral channel 0 input signal (RMT_SIG_IN0_IDX, signal index 83) to GPIO 15, please follow the steps below. Note that GPIO 15 is also named the MTDO pin:

1. Set the GPIO_FUNC_83_IN_SEL_CFG register field GPIO_FUNC83_IN_SEL value to 15.
2. As this is an input-only signal, set GPIO_FUNC15_OEN_SEL bit in GPIO_FUNC15_OUT_SEL_CFG_REG.
3. Clear bit 15 of GPIO_ENABLE_REG (field GPIO_ENABLE_DATA[15]).
4. Set the IO_MUX_GPIO15 register MCU_SEL field to 2 (GPIO function) and also set the FUN_IE bit (input mode).

4.2.3 Simple GPIO Input

The GPIO_IN_REG/GPIO_IN1_REG register holds the input values of each GPIO pad.

The input value of any GPIO pin can be read at any time without configuring the GPIO Matrix for a particular peripheral signal. However, it is necessary to enable the input in the IO_MUX by setting the FUN_IE bit in the IO_MUX_*x*_REG register corresponding to pad *X*, as mentioned in Section 4.2.2.

4.3 Peripheral Output via GPIO Matrix

4.3.1 Summary

To output a signal from a peripheral via the GPIO Matrix, the GPIO Matrix is configured to route the peripheral output signal (0-18, 23-37, 61-121, 140-125, 224-228) to one of the 28 GPIOs (0-19, 21-23, 25-27, 32-33).

The output signal is routed from the peripheral into the GPIO Matrix. It is then routed into the IO_MUX, which is configured to set the chosen pad to ”GPIO” function. This causes the output GPIO signal to be connected to the pad.

> **Note:**
> The peripheral output signals 224 to 228 can be configured to be routed in from one GPIO and output directly from another GPIO.

4.3.2 Functional Description

One of the 176 output signals can be selected to go through the GPIO matrix into the IO_MUX and then to a pad. Figure 9 illustrates the configuration.

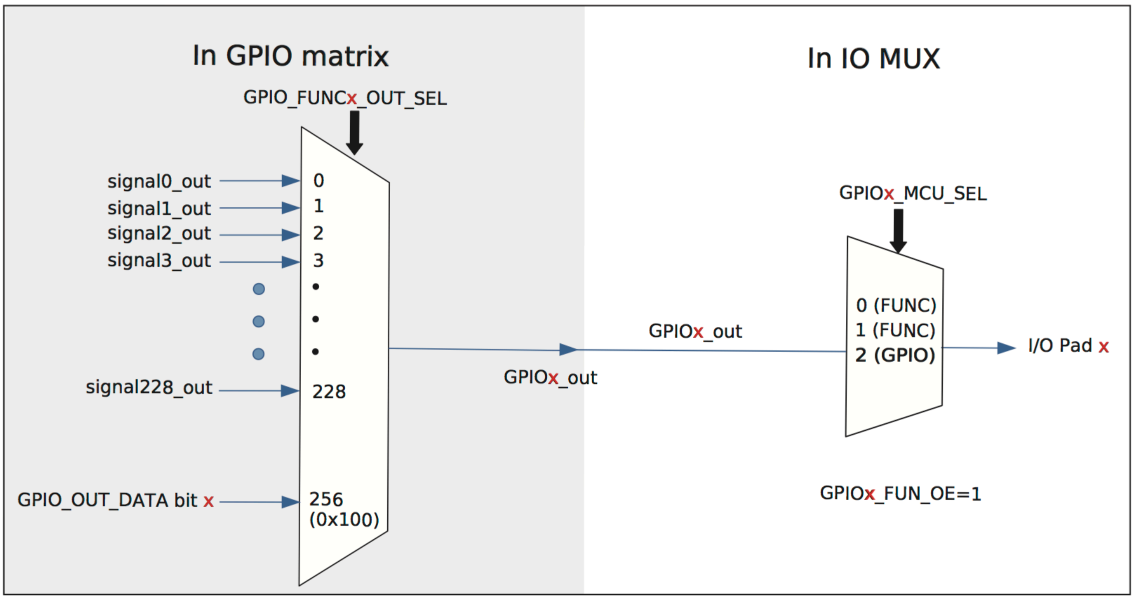


Figure 9: Output via GPIO Matrix

To output peripheral signal *Y* to particular GPIO pad *X*, follow these steps:

1. Configure the GPIO_FUNC*x*_OUT_SEL_CFG register and GPIO_ENABLE_DATA[*x*] field corresponding to GPIO *X* in the GPIO Matrix:
   - Set the GPIO_FUNC*x*_OUT_SEL field in GPIO_FUNC*x*_OUT_SEL_CFG to the numeric index (*Y*) of desired peripheral output signal *Y*.
   - If the signal should always be enabled as an output, set the GPIO_FUNC*x*_OEN_SEL bit in the GPIO_FUNC*x*_OUT_SEL_CFG register and the GPIO_ENABLE_DATA[*x*] field in the GPIO_ENABLE_REG register corresponding to GPIO pad *X*. To have the output enable signal decided by internal logic, clear the GPIO_FUNC*x*_OEN_SEL bit instead.
   - The GPIO_ENABLE_DATA[*x*] field is a bit in either GPIO_ENABLE_REG (GPIOs 0-31) or GPIO_ENABLE1_REG (GPIOs 32-39). Clear this bit to disable the output driver for the GPIO pad.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in the GPIO_PIN*x* register corresponding to GPIO pad *X*. For push/pull mode (default), clear this bit.
3. Configure the IO_MUX to select the GPIO Matrix. Set the IO_MUX_*x*_REG register corresponding to GPIO pad *X* as follows:
   - Set the function field (IO_*x*_MCU_SEL) to the IO_MUX function corresponding to GPIO *X* (this is Function #3—numeric value 2—for all pins).
   - Set the FUN_DRV field to the desired value for output strength (1-3). The higher the drive strength, the more current can be sourced/sunk from the pin.
   - If using open drain mode, set/clear the FUNC_WPU and FUNC_WPD bits to enable/disable the internal pull-up/down resistors.

Notes:

- The output signal from a single peripheral can be sent to multiple pads simultaneously.
- Only the 28 GPIOs can be used as outputs.
- The output signal can be inverted by setting the GPIO_FUNC*x*_OUT_INV_SEL bit.

4.3.3 Simple GPIO Output

The GPIO Matrix can also be used for simple GPIO output – setting a bit in the GPIO_OUT_DATA register will write to the corresponding GPIO pad.

To configure a pad as simple GPIO output, the GPIO Matrix GPIO_FUNC*x*_OUT_SEL register is configured with a special peripheral index value (0x100).

4.4 Direct I/O via IO_MUX

4.4.1 Summary

Some high speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency digital performance. In this case, the IO_MUX is used to connect these pads directly to the peripheral.

Selecting this option is less flexible than using the GPIO Matrix, as the IO_MUX register for each GPIO pad can only select from a limited number of functions. However, better high-frequency digital performance will be maintained.

4.4.2 Functional Description

Two registers must be configured in order to bypass the GPIO Matrix for peripheral I/O:

1. IO_MUX for the GPIO pad must be set to the required pad function. (Please refer to section 4.10 for a list of pad functions.)
2. For inputs, the SIG_IN_SEL register must be set to route the input directly to the peripheral.

4.5 RTC IO_MUX for Low Power and Analog I/O

4.5.1 Summary

18 GPIO pads have low power capabilities (RTC domain) and analog functions which are handled by the RTC subsystem of ESP32. The IO_MUX and GPIO Matrix are not used for these functions; rather, the RTC_MUX is used to redirect the I/O to the RTC subsystem.

When configured as RTC GPIOs, the output pads can still retain the output level value when the chip is in Deep-sleep mode, and the input pads can wake up the chip from Deep-sleep.

Section 4.11 has a list of RTC_MUX pins and their functions.

4.5.2 Functional Description

Each pad with analog and RTC functions is controlled by the RTC_IO_TOUCH_PAD*x*_TO_GPIO bit in the RTC_GPIO_PIN*x* register. By default this bit is set to 1, routing all I/O via the IO_MUX subsystem as described in earlier subsections.

If the RTC_IO_TOUCH_PAD*x*_TO_GPIO bit is cleared, then I/O to and from that pad is routed to the RTC subsystem. In this mode, the RTC_GPIO_PIN*x* register is used for digital I/O and the analog features of the pad are also available. See Section 4.11 for a list of RTC pin functions.

See 4.11 for a table mapping GPIO pads to their RTC equivalent pins and analog functions. Note that the RTC_IO_PIN*x* registers use the RTC GPIO pin numbering, not the GPIO pad numbering.

4.6 Light-sleep Mode Pin Functions

Pins can have different functions when the ESP32 is in Light-sleep mode. If the GPIO*xx*_SLP_SEL bit in the IO_MUX register for a GPIO pad is set to 1, a different set of registers is used to control the pad when the ESP32 is in Light-sleep mode:

Table 16: IO_MUX Light-sleep Pin Function Registers

IO_MUX Function	Normal Execution OR GPIO*xx*_SLP_SEL = 0	Light-sleep Mode AND GPIO*xx*_SLP_SEL = 1
Output Drive Strength	GPIO*xx*_FUNC_DRV	GPIO*xx*_MCU_DRV
Pullup Resistor	GPIO*xx*_FUNC_WPU	GPIO*xx*_MCU_WPU
Pulldown Resistor	GPIO*xx*_FUNC_WPD	GPIO*xx*_MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)	GPIO*xx*_MCU_OE

If GPIO*xx*_SLP_SEL is set to 0, the pin functions remain the same in both normal execution and Light-sleep mode.

4.7 Pad Hold Feature

Each IO pad (including the RTC pads) has an individual hold function controlled by a RTC register. When the pad is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO_MUX configuration or GPIO configuration is modified. Users can use the hold function for the pads to retain the pad state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

Note:

- For digital pads, to maintain the pad's input/output status in Deep-sleep mode, you can set REG_DG_PAD_FORCE_UNHOLD to 0 before powering down.
  For RTC pads, the input and output values are controlled by the corresponding bits of register RTC_CNTL_HOLD_FORCE_REG, and you can set it to 1 to hold the value or set it to 0 to unhold the value.
- For digital pads, to disable the hold function after the chip is woken up, you can set REG_DG_PAD_FORCE_UNHOLD to 1. To maintain the hold function of the pad, you can change the corresponding bit in the register by setting RTC_CNTL_HOLD_FORCE_REG to 1.

4.8 I/O Pad Power Supplies

Figure 10 and 11 show the IO pad power supplies.

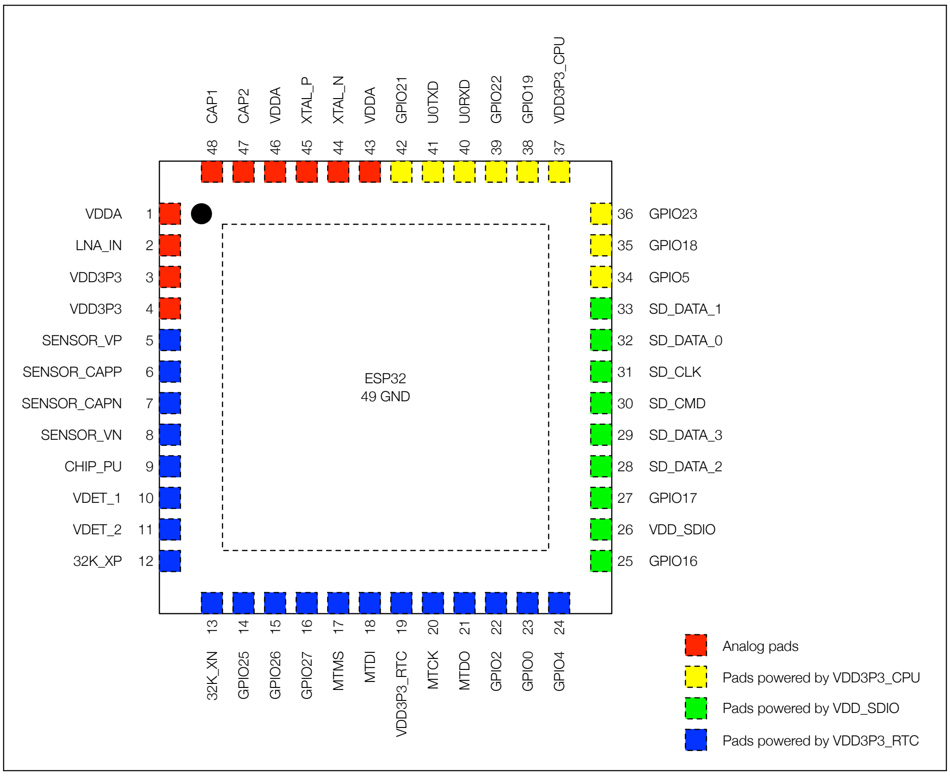


Figure 10: ESP32 I/O Pad Power Sources (QFN 6*6, Top View)

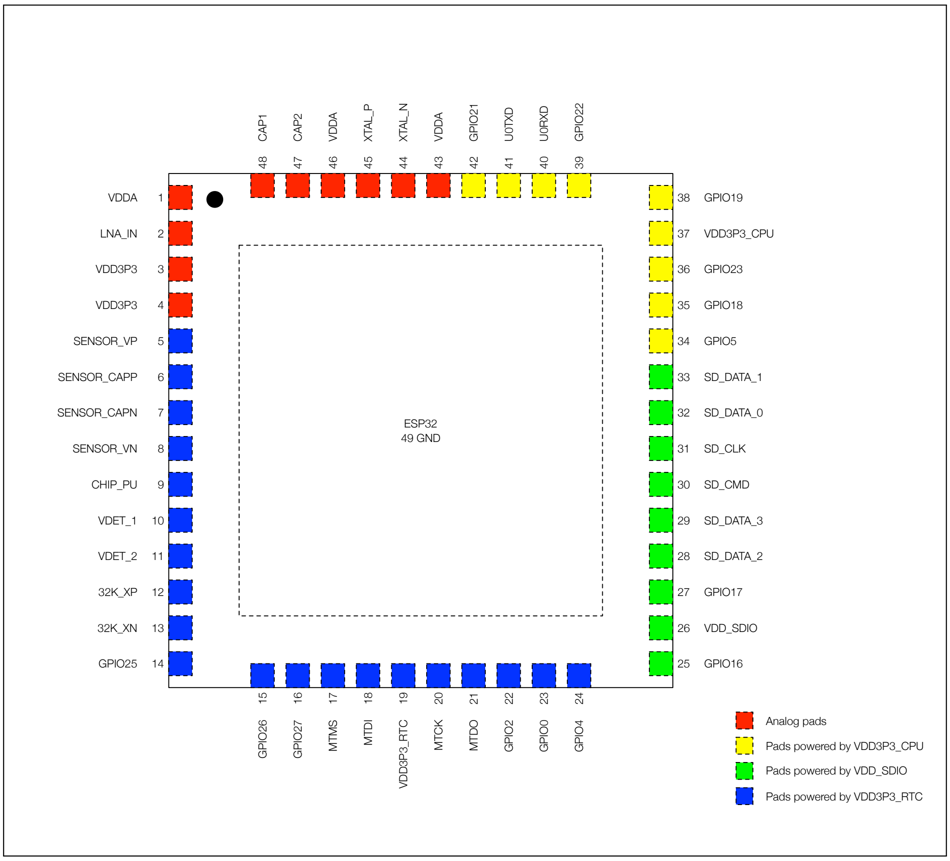


Figure 11: ESP32 I/O Pad Power Sources (QFN 5*5, Top View)

- Pads marked blue are RTC pads that have their individual analog function and can also act as normal digital IO pads. For details, please see Section 4.11.
- Pads marked yellow and green have digital functions only.
- Pads marked green can be powered externally or internally via VDD_SDIO (see below).

4.8.1 VDD_SDIO Power Domain

VDD_SDIO can source or sink current, allowing this power domain to be powered externally or internally. To power VDD_SDIO externally, apply the same power supply of VDD3P3_RTC to the VDD_SDIO pad.

Without an external power supply, the internal regulator will supply VDD_SDIO. The VDD_SDIO voltage can be configured to be either 1.8V or the same as VDD3P3_RTC, depending on the state of the MTDI pad at reset – a high level configures 1.8V and a low level configures the voltage to be the same as VDD3P3_RTC. Setting the efuse bit determines the default voltage of the VDD_SDIO. In addition, software can change the voltage of the VDD_SDIO by configuring register bits.

4.9 Peripheral Signal List

Table 17 contains a list of Peripheral Input/Output signals used by the GPIO Matrix:

Table 17: GPIO Matrix Peripheral Signals

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
0	SPICLK_in	SPICLK_out	YES
1	SPIQ_in	SPIQ_out	YES
2	SPID_in	SPID_out	YES
3	SPIHD_in	SPIHD_out	YES
4	SPIWP_in	SPIWP_out	YES
5	SPICS0_in	SPICS0_out	YES
6	SPICS1_in	SPICS1_out	-
7	SPICS2_in	SPICS2_out	-
8	HSPICLK_in	HSPICLK_out	YES
9	HSPIQ_in	HSPIQ_out	YES
10	HSPID_in	HSPID_out	YES
11	HSPICS0_in	HSPICS0_out	YES
12	HSPIHD_in	HSPIHD_out	YES
13	HSPIWP_in	HSPIWP_out	YES
14	U0RXD_in	U0TXD_out	YES
15	U0CTS_in	U0RTS_out	YES
16	U0DSR_in	U0DTR_out	-
17	U1RXD_in	U1TXD_out	YES
18	U1CTS_in	U1RTS_out	YES
23	I2S0O_BCK_in	I2S0O_BCK_out	-
24	I2S1O_BCK_in	I2S1O_BCK_out	-
25	I2S0O_WS_in	I2S0O_WS_out	-
26	I2S1O_WS_in	I2S1O_WS_out	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
27	I2S0I_BCK_in	I2S0I_BCK_out	-
28	I2S0I_WS_in	I2S0I_WS_out	-
29	I2CEXT0_SCL_in	I2CEXT0_SCL_out	-
30	I2CEXT0_SDA_in	I2CEXT0_SDA_out	-
31	pwm0_sync0_in	sdio_tohost_int_out	-
32	pwm0_sync1_in	pwm0_out0a	-
33	pwm0_sync2_in	pwm0_out0b	-
34	pwm0_f0_in	pwm0_out1a	-
35	pwm0_f1_in	pwm0_out1b	-
36	pwm0_f2_in	pwm0_out2a	-
37	-	pwm0_out2b	-
39	pcnt_sig_ch0_in0	-	-
40	pcnt_sig_ch1_in0	-	-
41	pcnt_ctrl_ch0_in0	-	-
42	pcnt_ctrl_ch1_in0	-	-
43	pcnt_sig_ch0_in1	-	-
44	pcnt_sig_ch1_in1	-	-
45	pcnt_ctrl_ch0_in1	-	-
46	pcnt_ctrl_ch1_in1	-	-
47	pcnt_sig_ch0_in2	-	-
48	pcnt_sig_ch1_in2	-	-
49	pcnt_ctrl_ch0_in2	-	-
50	pcnt_ctrl_ch1_in2	-	-
51	pcnt_sig_ch0_in3	-	-
52	pcnt_sig_ch1_in3	-	-
53	pcnt_ctrl_ch0_in3	-	-
54	pcnt_ctrl_ch1_in3	-	-
55	pcnt_sig_ch0_in4	-	-
56	pcnt_sig_ch1_in4	-	-
57	pcnt_ctrl_ch0_in4	-	-
58	pcnt_ctrl_ch1_in4	-	-
61	HSPICS1_in	HSPICS1_out	-
62	HSPICS2_in	HSPICS2_out	-
63	VSPICLK_in	VSPICLK_out_mux	YES
64	VSPIQ_in	VSPIQ_out	YES
65	VSPID_in	VSPID_out	YES
66	VSPIHD_in	VSPIHD_out	YES
67	VSPIWP_in	VSPIWP_out	YES
68	VSPICS0_in	VSPICS0_out	YES
69	VSPICS1_in	VSPICS1_out	-
70	VSPICS2_in	VSPICS2_out	-
71	pcnt_sig_ch0_in5	ledc_hs_sig_out0	-
72	pcnt_sig_ch1_in5	ledc_hs_sig_out1	-
73	pcnt_ctrl_ch0_in5	ledc_hs_sig_out2	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
74	pcnt_ctrl_ch1_in5	ledc_hs_sig_out3	-
75	pcnt_sig_ch0_in6	ledc_hs_sig_out4	-
76	pcnt_sig_ch1_in6	ledc_hs_sig_out5	-
77	pcnt_ctrl_ch0_in6	ledc_hs_sig_out6	-
78	pcnt_ctrl_ch1_in6	ledc_hs_sig_out7	-
79	pcnt_sig_ch0_in7	ledc_ls_sig_out0	-
80	pcnt_sig_ch1_in7	ledc_ls_sig_out1	-
81	pcnt_ctrl_ch0_in7	ledc_ls_sig_out2	-
82	pcnt_ctrl_ch1_in7	ledc_ls_sig_out3	-
83	rmt_sig_in0	ledc_ls_sig_out4	-
84	rmt_sig_in1	ledc_ls_sig_out5	-
85	rmt_sig_in2	ledc_ls_sig_out6	-
86	rmt_sig_in3	ledc_ls_sig_out7	-
87	rmt_sig_in4	rmt_sig_out0	-
88	rmt_sig_in5	rmt_sig_out1	-
89	rmt_sig_in6	rmt_sig_out2	-
90	rmt_sig_in7	rmt_sig_out3	-
91	-	rmt_sig_out4	-
92	-	rmt_sig_out5	-
93	-	rmt_sig_out6	-
94	-	rmt_sig_out7	-
95	I2CEXT1_SCL_in	I2CEXT1_SCL_out	-
96	I2CEXT1_SDA_in	I2CEXT1_SDA_out	-
97	host_card_detect_n_1	host_ccmd_od_pullup_en_n	-
98	host_card_detect_n_2	host_rst_n_1	-
99	host_card_write_prt_1	host_rst_n_2	-
100	host_card_write_prt_2	gpio_sd0_out	-
101	host_card_int_n_1	gpio_sd1_out	-
102	host_card_int_n_2	gpio_sd2_out	-
103	pwm1_sync0_in	gpio_sd3_out	-
104	pwm1_sync1_in	gpio_sd4_out	-
105	pwm1_sync2_in	gpio_sd5_out	-
106	pwm1_f0_in	gpio_sd6_out	-
107	pwm1_f1_in	gpio_sd7_out	-
108	pwm1_f2_in	pwm1_out0a	-
109	pwm0_cap0_in	pwm1_out0b	-
110	pwm0_cap1_in	pwm1_out1a	-
111	pwm0_cap2_in	pwm1_out1b	-
112	pwm1_cap0_in	pwm1_out2a	-
113	pwm1_cap1_in	pwm1_out2b	-
114	pwm1_cap2_in	pwm2_out1h	-
115	pwm2_flta	pwm2_out1l	-
116	pwm2_fltb	pwm2_out2h	-
117	pwm2_cap1_in	pwm2_out2l	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
118	pwm2_cap2_in	pwm2_out3h	-
119	pwm2_cap3_in	pwm2_out3l	-
120	pwm3_flta	pwm2_out4h	-
121	pwm3_fltb	pwm2_out4l	-
122	pwm3_cap1_in	-	-
123	pwm3_cap2_in	-	-
124	pwm3_cap3_in	-	-
140	I2S0I_DATA_in0	I2S0O_DATA_out0	-
141	I2S0I_DATA_in1	I2S0O_DATA_out1	-
142	I2S0I_DATA_in2	I2S0O_DATA_out2	-
143	I2S0I_DATA_in3	I2S0O_DATA_out3	-
144	I2S0I_DATA_in4	I2S0O_DATA_out4	-
145	I2S0I_DATA_in5	I2S0O_DATA_out5	-
146	I2S0I_DATA_in6	I2S0O_DATA_out6	-
147	I2S0I_DATA_in7	I2S0O_DATA_out7	-
148	I2S0I_DATA_in8	I2S0O_DATA_out8	-
149	I2S0I_DATA_in9	I2S0O_DATA_out9	-
150	I2S0I_DATA_in10	I2S0O_DATA_out10	-
151	I2S0I_DATA_in11	I2S0O_DATA_out11	-
152	I2S0I_DATA_in12	I2S0O_DATA_out12	-
153	I2S0I_DATA_in13	I2S0O_DATA_out13	-
154	I2S0I_DATA_in14	I2S0O_DATA_out14	-
155	I2S0I_DATA_in15	I2S0O_DATA_out15	-
156	-	I2S0O_DATA_out16	-
157	-	I2S0O_DATA_out17	-
158	-	I2S0O_DATA_out18	-
159	-	I2S0O_DATA_out19	-
160	-	I2S0O_DATA_out20	-
161	-	I2S0O_DATA_out21	-
162	-	I2S0O_DATA_out22	-
163	-	I2S0O_DATA_out23	-
164	I2S1I_BCK_in	I2S1I_BCK_out	-
165	I2S1I_WS_in	I2S1I_WS_out	-
166	I2S1I_DATA_in0	I2S1O_DATA_out0	-
167	I2S1I_DATA_in1	I2S1O_DATA_out1	-
168	I2S1I_DATA_in2	I2S1O_DATA_out2	-
169	I2S1I_DATA_in3	I2S1O_DATA_out3	-
170	I2S1I_DATA_in4	I2S1O_DATA_out4	-
171	I2S1I_DATA_in5	I2S1O_DATA_out5	-
172	I2S1I_DATA_in6	I2S1O_DATA_out6	-
173	I2S1I_DATA_in7	I2S1O_DATA_out7	-
174	I2S1I_DATA_in8	I2S1O_DATA_out8	-
175	I2S1I_DATA_in9	I2S1O_DATA_out9	-
176	I2S1I_DATA_in10	I2S1O_DATA_out10	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
177	I2S1I_DATA_in11	I2S1O_DATA_out11	-
178	I2S1I_DATA_in12	I2S1O_DATA_out12	-
179	I2S1I_DATA_in13	I2S1O_DATA_out13	-
180	I2S1I_DATA_in14	I2S1O_DATA_out14	-
181	I2S1I_DATA_in15	I2S1O_DATA_out15	-
182	-	I2S1O_DATA_out16	-
183	-	I2S1O_DATA_out17	-
184	-	I2S1O_DATA_out18	-
185	-	I2S1O_DATA_out19	-
186	-	I2S1O_DATA_out20	-
187	-	I2S1O_DATA_out21	-
188	-	I2S1O_DATA_out22	-
189	-	I2S1O_DATA_out23	-
190	I2S0I_H_SYNC	pwm3_out1h	-
191	I2S0I_V_SYNC	pwm3_out1l	-
192	I2S0I_H_ENABLE	pwm3_out2h	-
193	I2S1I_H_SYNC	pwm3_out2l	-
194	I2S1I_V_SYNC	pwm3_out3h	-
195	I2S1I_H_ENABLE	pwm3_out3l	-
196	-	pwm3_out4h	-
197	-	pwm3_out4l	-
198	U2RXD_in	U2TXD_out	YES
199	U2CTS_in	U2RTS_out	YES
200	emac_mdc_i	emac_mdc_o	-
201	emac_mdi_i	emac_mdo_o	-
202	emac_crs_i	emac_crs_o	-
203	emac_col_i	emac_col_o	-
204	pcmfsync_in	bt_audio0_irq	-
205	pcmclk_in	bt_audio1_irq	-
206	pcmdin	bt_audio2_irq	-
207	-	ble_audio0_irq	-
208	-	ble_audio1_irq	-
209	-	ble_audio2_irq	-
210	-	pcmfsync_out	-
211	-	pcmclk_out	-
212	-	pcmdout	-
213	-	ble_audio_sync0_p	-
214	-	ble_audio_sync1_p	-
215	-	ble_audio_sync2_p	-
224	-	sig_in_func224	-
225	-	sig_in_func225	-
226	-	sig_in_func226	-
227	-	sig_in_func227	-
228	-	sig_in_func228	-

Direct I/O in IO_MUX ”YES” means that this signal is also available directly via IO_MUX. To apply the GPIO Matrix to these signals, their corresponding SIG_IN_SEL register must be cleared.

4.10 IO_MUX Pad List

Table 18 shows the IO_MUX functions for each I/O pad:

Table 18: IO_MUX Pad Summary

GPIO	Pad Name	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Reset	Notes
0	GPIO0	GPIO0	CLK_OUT1	GPIO0	-	-	EMAC_TX_CLK	3	R
1	U0TXD	U0TXD	CLK_OUT3	GPIO1	-	-	EMAC_RXD2	3	-
2	GPIO2	GPIO2	HSPIWP	GPIO2	HS2_DATA0	SD_DATA0	-	2	R
3	U0RXD	U0RXD	CLK_OUT2	GPIO3	-	-	-	3	-
4	GPIO4	GPIO4	HSPIHD	GPIO4	HS2_DATA1	SD_DATA1	EMAC_TX_ER	2	R
5	GPIO5	GPIO5	VSPICS0	GPIO5	HS1_DATA6	-	EMAC_RX_CLK	3	-
6	SD_CLK	SD_CLK	SPICLK	GPIO6	HS1_CLK	U1CTS	-	3	-
7	SD_DATA_0	SD_DATA0	SPIQ	GPIO7	HS1_DATA0	U2RTS	-	3	-
8	SD_DATA_1	SD_DATA1	SPID	GPIO8	HS1_DATA1	U2CTS	-	3	-
9	SD_DATA_2	SD_DATA2	SPIHD	GPIO9	HS1_DATA2	U1RXD	-	3	-
10	SD_DATA_3	SD_DATA3	SPIWP	GPIO10	HS1_DATA3	U1TXD	-	3	-
11	SD_CMD	SD_CMD	SPICS0	GPIO11	HS1_CMD	U1RTS	-	3	-
12	MTDI	MTDI	HSPIQ	GPIO12	HS2_DATA2	SD_DATA2	EMAC_TXD3	2	R
13	MTCK	MTCK	HSPID	GPIO13	HS2_DATA3	SD_DATA3	EMAC_RX_ER	1	R
14	MTMS	MTMS	HSPICLK	GPIO14	HS2_CLK	SD_CLK	EMAC_TXD2	1	R
15	MTDO	MTDO	HSPICS0	GPIO15	HS2_CMD	SD_CMD	EMAC_RXD3	3	R
16	GPIO16	GPIO16	-	GPIO16	HS1_DATA4	U2RXD	EMAC_CLK_OUT	1	-
17	GPIO17	GPIO17	-	GPIO17	HS1_DATA5	U2TXD	EMAC_CLK_180	1	-
18	GPIO18	GPIO18	VSPICLK	GPIO18	HS1_DATA7	-	-	1	-
19	GPIO19	GPIO19	VSPIQ	GPIO19	U0CTS	-	EMAC_TXD0	1	-
21	GPIO21	GPIO21	VSPIHD	GPIO21	-	-	EMAC_TX_EN	1	-
22	GPIO22	GPIO22	VSPIWP	GPIO22	U0RTS	-	EMAC_TXD1	1	-
23	GPIO23	GPIO23	VSPID	GPIO23	HS1_STROBE	-	-	1	-
25	GPIO25	GPIO25	-	GPIO25	-	-	EMAC_RXD0	0	R
26	GPIO26	GPIO26	-	GPIO26	-	-	EMAC_RXD1	0	R
27	GPIO27	GPIO27	-	GPIO27	-	-	EMAC_RX_DV	1	R
32	32K_XP	GPIO32	-	GPIO32	-	-	-	0	R
33	32K_XN	GPIO33	-	GPIO33	-	-	-	0	R
34	VDET_1	GPIO34	-	GPIO34	-	-	-	0	R, I
35	VDET_2	GPIO35	-	GPIO35	-	-	-	0	R, I
36	SENSOR_VP	GPIO36	-	GPIO36	-	-	-	0	R, I
37	SENSOR_CAPP	GPIO37	-	GPIO37	-	-	-	0	R, I
38	SENSOR_CAPN	GPIO38	-	GPIO38	-	-	-	0	R, I
39	SENSOR_VN	GPIO39	-	GPIO39	-	-	-	0	R, I

Reset Configurations

”Reset” column shows each pad’s default configurations after reset:

- **0** - IE=0 (input disabled).
- **1** - IE=1 (input enabled).
- **2** - IE=1, WPD=1 (input enabled, pulldown resistor).
- **3** - IE=1, WPU=1 (input enabled, pullup resistor).

Notes

- **R** - Pad has RTC/analog functions via RTC_MUX.
- **I** - Pad can only be configured as input GPIO.

Please refer to the ESP32 Pin Lists in *ESP32 Datasheet* for more details.

4.11 RTC_MUX Pin List

Table 19 shows the RTC pins and how they correspond to GPIO pads:

Table 19: RTC_MUX Pin Summary

RTC GPIO Num	GPIO Num	Pad Name	Analog Function		
			1	2	3
0	36	SENSOR_VP	ADC_H	ADC1_CH0	-
1	37	SENSOR_CAPP	ADC_H	ADC1_CH1	-
2	38	SENSOR_CAPN	ADC_H	ADC1_CH2	-
3	39	SENSOR_VN	ADC_H	ADC1_CH3	-
4	34	VDET_1	-	ADC1_CH6	-
5	35	VDET_2	-	ADC1_CH7	-
6	25	GPIO25	DAC_1	ADC2_CH8	-
7	26	GPIO26	DAC_2	ADC2_CH9	-
8	33	32K_XN	XTAL_32K_N	ADC1_CH5	TOUCH8
9	32	32K_XP	XTAL_32K_P	ADC1_CH4	TOUCH9
10	4	GPIO4	-	ADC2_CH0	TOUCH0
11	0	GPIO0	-	ADC2_CH1	TOUCH1
12	2	GPIO2	-	ADC2_CH2	TOUCH2
13	15	MTDO	-	ADC2_CH3	TOUCH3
14	13	MTCK	-	ADC2_CH4	TOUCH4
15	12	MTDI	-	ADC2_CH5	TOUCH5
16	14	MTMS	-	ADC2_CH6	TOUCH6
17	27	GPIO27	-	ADC2_CH7	TOUCH7

4.12 Register Summary

Name	Description	Address	Access
GPIO_OUT_REG	GPIO 0-31 output register	0x3FF44004	R/W
GPIO_OUT_W1TS_REG	GPIO 0-31 output register_W1TS	0x3FF44008	WO
GPIO_OUT_W1TC_REG	GPIO 0-31 output register_W1TC	0x3FF4400C	WO
GPIO_OUT1_REG	GPIO 32-39 output register	0x3FF44010	R/W
GPIO_OUT1_W1TS_REG	GPIO 32-39 output bit set register	0x3FF44014	WO
GPIO_OUT1_W1TC_REG	GPIO 32-39 output bit clear register	0x3FF44018	WO
GPIO_ENABLE_REG	GPIO 0-31 output enable register	0x3FF44020	R/W
GPIO_ENABLE_W1TS_REG	GPIO 0-31 output enable register_W1TS	0x3FF44024	WO
GPIO_ENABLE_W1TC_REG	GPIO 0-31 output enable register_W1TC	0x3FF44028	WO
GPIO_ENABLE1_REG	GPIO 32-39 output enable register	0x3FF4402C	R/W

Name	Description	Address	Access
GPIO_ENABLE1_W1TS_REG	GPIO 32-39 output enable bit set register	0x3FF44030	WO
GPIO_ENABLE1_W1TC_REG	GPIO 32-39 output enable bit clear register	0x3FF44034	WO
GPIO_STRAP_REG	Bootstrap pin value register	0x3FF44038	RO
GPIO_IN_REG	GPIO 0-31 input register	0x3FF4403C	RO
GPIO_IN1_REG	GPIO 32-39 input register	0x3FF44040	RO
GPIO_STATUS_REG	GPIO 0-31 interrupt status register	0x3FF44044	R/W
GPIO_STATUS_W1TS_REG	GPIO 0-31 interrupt status register_W1TS	0x3FF44048	WO
GPIO_STATUS_W1TC_REG	GPIO 0-31 interrupt status register_W1TC	0x3FF4404C	WO
GPIO_STATUS1_REG	GPIO 32-39 interrupt status register1	0x3FF44050	R/W
GPIO_STATUS1_W1TS_REG	GPIO 32-39 interrupt status bit set register	0x3FF44054	WO
GPIO_STATUS1_W1TC_REG	GPIO 32-39 interrupt status bit clear register	0x3FF44058	WO
GPIO_ACPU_INT_REG	GPIO 0-31 APP_CPU interrupt status	0x3FF44060	RO
GPIO_ACPU_NMI_INT_REG	GPIO 0-31 APP_CPU non-maskable interrupt status	0x3FF44064	RO
GPIO_PCPU_INT_REG	GPIO 0-31 PRO_CPU interrupt status	0x3FF44068	RO
GPIO_PCPU_NMI_INT_REG	GPIO 0-31 PRO_CPU non-maskable interrupt status	0x3FF4406C	RO
GPIO_ACPU_INT1_REG	GPIO 32-39 APP_CPU interrupt status	0x3FF44074	RO
GPIO_ACPU_NMI_INT1_REG	GPIO 32-39 APP_CPU non-maskable interrupt status	0x3FF44078	RO
GPIO_PCPU_INT1_REG	GPIO 32-39 PRO_CPU interrupt status	0x3FF4407C	RO
GPIO_PCPU_NMI_INT1_REG	GPIO 32-39 PRO_CPU non-maskable interrupt status	0x3FF44080	RO
GPIO_PIN0_REG	Configuration for GPIO pin 0	0x3FF44088	R/W
GPIO_PIN1_REG	Configuration for GPIO pin 1	0x3FF4408C	R/W
GPIO_PIN2_REG	Configuration for GPIO pin 2	0x3FF44090	R/W
...	...		
GPIO_PIN38_REG	Configuration for GPIO pin 38	0x3FF44120	R/W
GPIO_PIN39_REG	Configuration for GPIO pin 39	0x3FF44124	R/W
GPIO_FUNC0_IN_SEL_CFG_REG	Peripheral function 0 input selection register	0x3FF44130	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Peripheral function 1 input selection register	0x3FF44134	R/W
...	...		
GPIO_FUNC254_IN_SEL_CFG_REG	Peripheral function 254 input selection register	0x3FF44528	R/W
GPIO_FUNC255_IN_SEL_CFG_REG	Peripheral function 255 input selection register	0x3FF4452C	R/W
GPIO_FUNC0_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 0	0x3FF44530	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 1	0x3FF44534	R/W
...	...		
GPIO_FUNC38_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 38	0x3FF445C8	R/W
GPIO_FUNC39_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 39	0x3FF445CC	R/W

Name	Description	Address	Access
IO_MUX_PIN_CTRL	Clock output configuration register	0x3FF49000	R/W
IO_MUX_GPIO36_REG	Configuration register for pad GPIO36	0x3FF49004	R/W
IO_MUX_GPIO37_REG	Configuration register for pad GPIO37	0x3FF49008	R/W

Name	Description	Address	Access
IO_MUX_GPIO38_REG	Configuration register for pad GPIO38	0x3FF4900C	R/W
IO_MUX_GPIO39_REG	Configuration register for pad GPIO39	0x3FF49010	R/W
IO_MUX_GPIO34_REG	Configuration register for pad GPIO34	0x3FF49014	R/W
IO_MUX_GPIO35_REG	Configuration register for pad GPIO35	0x3FF49018	R/W
IO_MUX_GPIO32_REG	Configuration register for pad GPIO32	0x3FF4901C	R/W
IO_MUX_GPIO33_REG	Configuration register for pad GPIO33	0x3FF49020	R/W
IO_MUX_GPIO25_REG	Configuration register for pad GPIO25	0x3FF49024	R/W
IO_MUX_GPIO26_REG	Configuration register for pad GPIO26	0x3FF49028	R/W
IO_MUX_GPIO27_REG	Configuration register for pad GPIO27	0x3FF4902C	R/W
IO_MUX_MTMS_REG	Configuration register for pad MTMS	0x3FF49030	R/W
IO_MUX_MTDI_REG	Configuration register for pad MTDI	0x3FF49034	R/W
IO_MUX_MTCK_REG	Configuration register for pad MTCK	0x3FF49038	R/W
IO_MUX_MTDO_REG	Configuration register for pad MTDO	0x3FF4903C	R/W
IO_MUX_GPIO2_REG	Configuration register for pad GPIO2	0x3FF49040	R/W
IO_MUX_GPIO0_REG	Configuration register for pad GPIO0	0x3FF49044	R/W
IO_MUX_GPIO4_REG	Configuration register for pad GPIO4	0x3FF49048	R/W
IO_MUX_GPIO16_REG	Configuration register for pad GPIO16	0x3FF4904C	R/W
IO_MUX_GPIO17_REG	Configuration register for pad GPIO17	0x3FF49050	R/W
IO_MUX_SD_DATA2_REG	Configuration register for pad SD_DATA2	0x3FF49054	R/W
IO_MUX_SD_DATA3_REG	Configuration register for pad SD_DATA3	0x3FF49058	R/W
IO_MUX_SD_CMD_REG	Configuration register for pad SD_CMD	0x3FF4905C	R/W
IO_MUX_SD_CLK_REG	Configuration register for pad SD_CLK	0x3FF49060	R/W
IO_MUX_SD_DATA0_REG	Configuration register for pad SD_DATA0	0x3FF49064	R/W
IO_MUX_SD_DATA1_REG	Configuration register for pad SD_DATA1	0x3FF49068	R/W
IO_MUX_GPIO5_REG	Configuration register for pad GPIO5	0x3FF4906C	R/W
IO_MUX_GPIO18_REG	Configuration register for pad GPIO18	0x3FF49070	R/W
IO_MUX_GPIO19_REG	Configuration register for pad GPIO19	0x3FF49074	R/W
IO_MUX_GPIO20_REG	Configuration register for pad GPIO20	0x3FF49078	R/W
IO_MUX_GPIO21_REG	Configuration register for pad GPIO21	0x3FF4907C	R/W
IO_MUX_GPIO22_REG	Configuration register for pad GPIO22	0x3FF49080	R/W
IO_MUX_U0RXD_REG	Configuration register for pad U0RXD	0x3FF49084	R/W
IO_MUX_U0TXD_REG	Configuration register for pad U0TXD	0x3FF49088	R/W
IO_MUX_GPIO23_REG	Configuration register for pad GPIO23	0x3FF4908C	R/W
IO_MUX_GPIO24_REG	Configuration register for pad GPIO24	0x3FF49090	R/W

Name	Description	Address	Access
GPIO configuration / data registers			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO output register	0x3FF48400	R/W
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register	0x3FF48404	WO
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register	0x3FF48408	WO
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO output enable register	0x3FF4840C	R/W
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit set register	0x3FF48410	WO
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register	0x3FF48414	WO
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register	0x3FF48418	WO

Name	Description	Address	Access
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register	0x3FF4841C	WO
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register	0x3FF48420	WO
RTCIO_RTC_GPIO_IN_REG	RTC GPIO input register	0x3FF48424	RO
RTCIO_RTC_GPIO_PIN0_REG	RTC configuration for pin 0	0x3FF48428	R/W
RTCIO_RTC_GPIO_PIN1_REG	RTC configuration for pin 1	0x3FF4842C	R/W
RTCIO_RTC_GPIO_PIN2_REG	RTC configuration for pin 2	0x3FF48430	R/W
RTCIO_RTC_GPIO_PIN3_REG	RTC configuration for pin 3	0x3FF48434	R/W
RTCIO_RTC_GPIO_PIN4_REG	RTC configuration for pin 4	0x3FF48438	R/W
RTCIO_RTC_GPIO_PIN5_REG	RTC configuration for pin 5	0x3FF4843C	R/W
RTCIO_RTC_GPIO_PIN6_REG	RTC configuration for pin 6	0x3FF48440	R/W
RTCIO_RTC_GPIO_PIN7_REG	RTC configuration for pin 7	0x3FF48444	R/W
RTCIO_RTC_GPIO_PIN8_REG	RTC configuration for pin 8	0x3FF48448	R/W
RTCIO_RTC_GPIO_PIN9_REG	RTC configuration for pin 9	0x3FF4844C	R/W
RTCIO_RTC_GPIO_PIN10_REG	RTC configuration for pin 10	0x3FF48450	R/W
RTCIO_RTC_GPIO_PIN11_REG	RTC configuration for pin 11	0x3FF48454	R/W
RTCIO_RTC_GPIO_PIN12_REG	RTC configuration for pin 12	0x3FF48458	R/W
RTCIO_RTC_GPIO_PIN13_REG	RTC configuration for pin 13	0x3FF4845C	R/W
RTCIO_RTC_GPIO_PIN14_REG	RTC configuration for pin 14	0x3FF48460	R/W
RTCIO_RTC_GPIO_PIN15_REG	RTC configuration for pin 15	0x3FF48464	R/W
RTCIO_RTC_GPIO_PIN16_REG	RTC configuration for pin 16	0x3FF48468	R/W
RTCIO_RTC_GPIO_PIN17_REG	RTC configuration for pin 17	0x3FF4846C	R/W
RTCIO_DIG_PAD_HOLD_REG	RTC GPIO hold register	0x3FF48474	R/W
GPIO RTC function configuration registers			
RTCIO_HALL_SENS_REG	Hall sensor configuration	0x3FF48478	R/W
RTCIO_SENSOR_PADS_REG	Sensor pads configuration register	0x3FF4847C	R/W
RTCIO_ADC_PAD_REG	ADC configuration register	0x3FF48480	R/W
RTCIO_PAD_DAC1_REG	DAC1 configuration register	0x3FF48484	R/W
RTCIO_PAD_DAC2_REG	DAC2 configuration register	0x3FF48488	R/W
RTCIO_XTAL_32K_PAD_REG	32KHz crystal pads configuration register	0x3FF4848C	R/W
RTCIO_TOUCH_CFG_REG	Touch sensor configuration register	0x3FF48490	R/W
RTCIO_TOUCH_PAD0_REG	Touch pad configuration register	0x3FF48494	R/W
...	...		
RTCIO_TOUCH_PAD9_REG	Touch pad configuration register	0x3FF484B8	R/W
RTCIO_EXT_WAKEUP0_REG	External wake up configuration register	0x3FF484BC	R/W
RTCIO_XTL_EXT_CTR_REG	Crystal power down enable GPIO source	0x3FF484C0	R/W
RTCIO_SAR_I2C_IO_REG	RTC I2C pad selection	0x3FF484C4	R/W

4.13 Registers

Register 4.1: GPIO_OUT_REG (0x0004)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_OUT_REG GPIO0-31 output value. (R/W)

Register 4.2: GPIO_OUT_W1TS_REG (0x0008)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_OUT_W1TS_REG GPIO0-31 output set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be set. (WO)

Register 4.3: GPIO_OUT_W1TC_REG (0x000c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_OUT_W1TC_REG GPIO0-31 output clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT_REG will be cleared. (WO)

Register 4.4: GPIO_OUT1_REG (0x0010)

(reserved)	GPIO_OUT_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_OUT_DATA GPIO32-39 output value. (R/W)

Register 4.5: GPIO_OUT1_W1TS_REG (0x0014)

(reserved)	GPIO_OUT_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_OUT_DATA GPIO32-39 output value set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be set. (WO)

Register 4.6: GPIO_OUT1_W1TC_REG (0x0018)

(reserved)	GPIO_OUT_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_OUT_DATA GPIO32-39 output value clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_OUT1_DATA will be cleared. (WO)

Register 4.7: GPIO_ENABLE_REG (0x0020)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ENABLE_REG GPIO0-31 output enable. (R/W)

Register 4.8: GPIO_ENABLE_W1TS_REG (0x0024)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ENABLE_W1TS_REG GPIO0-31 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be set. (WO)

Register 4.9: GPIO_ENABLE_W1TC_REG (0x0028)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ENABLE_W1TC_REG GPIO0-31 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE will be cleared. (WO)

Register 4.10: GPIO_ENABLE1_REG (0x002c)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable. (R/W)

Register 4.11: GPIO_ENABLE1_W1TS_REG (0x0030)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be set. (WO)

Register 4.12: GPIO_ENABLE1_W1TC_REG (0x0034)

(reserved)	GPIO_ENABLE_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_ENABLE_DATA GPIO32-39 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_ENABLE1 will be cleared. (WO)

Register 4.13: GPIO_STRAP_REG (0x0038)

(reserved)	GPIO_STRAPPING	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x x x x x x x x x	Reset

GPIO_STRAPPING GPIO strapping results: Bit5-bit0 of boot_sel_chip[5:0] correspond to MTDI, GPIO0, GPIO2, GPIO4, MTDO, GPIO5, respectively.

Register 4.14: GPIO_IN_REG (0x003c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_IN_REG GPIO0-31 input value. Each bit represents a pad input value, 1 for high level and 0 for low level. (RO)

Register 4.15: GPIO_IN1_REG (0x0040)

(reserved)	GPIO_IN_DATA_NEXT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_IN_DATA_NEXT GPIO32-39 input value. Each bit represents a pad input value. (RO)

Register 4.16: GPIO_STATUS_REG (0x0044)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_REG GPIO0-31 interrupt status register. Each bit can be either of the two interrupt sources for the two CPUs. The enable bits in GPIO_STATUS_INTERRUPT, corresponding to the 0-4 bits in GPIO_PIN*n*_REG should be set to 1. (R/W)

Register 4.17: GPIO_STATUS_W1TS_REG (0x0048)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_W1TS_REG GPIO0-31 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set. (WO)

Register 4.18: GPIO_STATUS_W1TC_REG (0x004c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_W1TC_REG GPIO0-31 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. (WO)

Register 4.19: GPIO_STATUS1_REG (0x0050)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status. (R/W)

Register 4.20: GPIO_STATUS1_W1TS_REG (0x0054)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status set register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT1 will be set. (WO)

Register 4.21: GPIO_STATUS1_W1TC_REG (0x0058)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 interrupt status clear register. For every bit that is 1 in the value written here, the corresponding bit in GPIO_STATUS_INTERRUPT1 will be cleared. (WO)

Register 4.22: GPIO_ACPU_INT_REG (0x0060)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ACPU_INT_REG GPIO0-31 APP CPU interrupt status. (RO)

Register 4.23: GPIO_ACPU_NMI_INT_REG (0x0064)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ACPU_NMI_INT_REG GPIO0-31 APP CPU non-maskable interrupt status. (RO)

Register 4.24: GPIO_PCPU_INT_REG (0x0068)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_INT_REG GPIO0-31 PRO CPU interrupt status. (RO)

Register 4.25: GPIO_PCPU_NMI_INT_REG (0x006c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_NMI_INT_REG GPIO0-31 PRO CPU non-maskable interrupt status. (RO)

Register 4.26: GPIO_ACPU_INT1_REG (0x0074)

(reserved)	GPIO_APPCPU_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_APPCPU_INT GPIO32-39 APP CPU interrupt status. (RO)

Register 4.27: GPIO_ACPU_NMI_INT1_REG (0x0078)

(reserved)	GPIO_APPCPU_NMI_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_APPCPU_NMI_INT GPIO32-39 APP CPU non-maskable interrupt status. (RO)

Register 4.28: GPIO_PCPU_INT1_REG (0x007c)

(reserved)		GPIO_PROCPU_INT		
31	8	7	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		x x x x x x x x		Reset

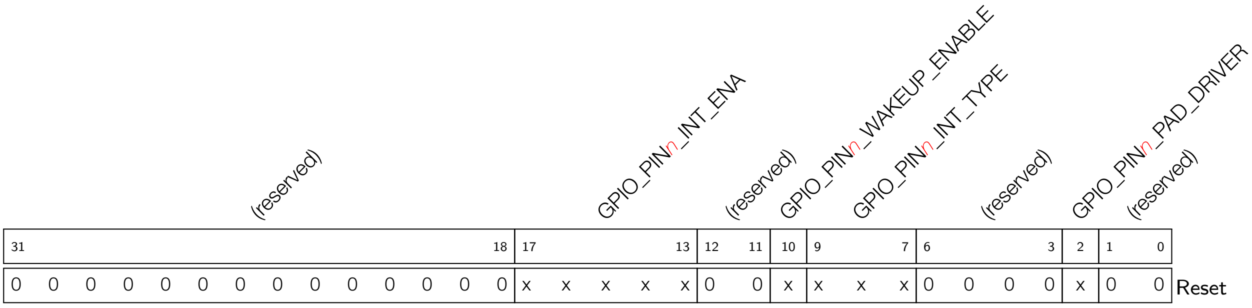
GPIO_PROCPU_INT GPIO32-39 PRO CPU interrupt status. (RO)

Register 4.29: GPIO_PCPU_NMI_INT1_REG (0x0080)

(reserved)		GPIO_PROCPU_NMI_INT		
31	8	7	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		x x x x x x x x		Reset

GPIO_PROCPU_NMI_INT GPIO32-39 PRO CPU non-maskable interrupt status. (RO)

Register 4.30: GPIO_PIN*n*_REG (*n*: 0-39) (0x88+0x4**n*)



GPIO_PIN*n*_INT_ENA Interrupt enable bits for pin *n*: (R/W)

bit0: APP CPU interrupt enable;

bit1: APP CPU non-maskable interrupt enable;

bit3: PRO CPU interrupt enable;

bit4: PRO CPU non-maskable interrupt enable.

GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable will only wake up the CPU from Light-sleep. (R/W)

GPIO_PIN*n*_INT_TYPE Interrupt type selection: (R/W)

0: GPIO interrupt disable;

1: rising edge trigger;

2: falling edge trigger;

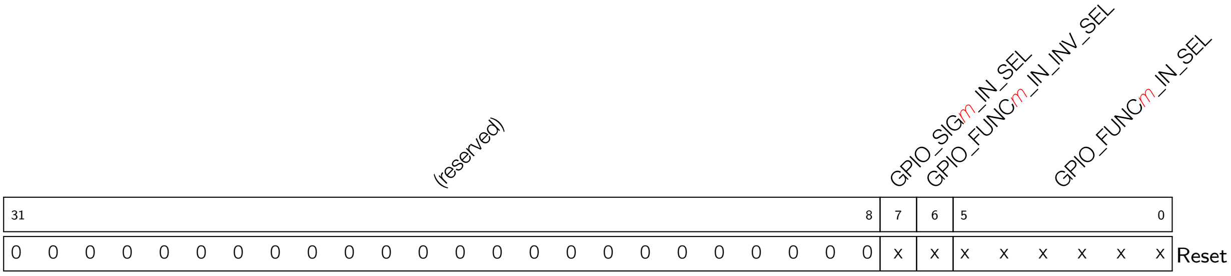
3: any edge trigger;

4: low level trigger;

5: high level trigger.

GPIO_PIN*n*_PAD_DRIVER 0: normal output; 1: open drain output. (R/W)

Register 4.31: GPIO_FUNC*m*_IN_SEL_CFG_REG (*m*: 0-255) (0x130+0x4**m*)



GPIO_SIG*m*_IN_SEL Bypass the GPIO Matrix. 1: route through GPIO Matrix, 0: connect signal directly to peripheral configured in the IO_MUX. (R/W)

GPIO_FUNC*m*_IN_INV_SEL Invert the input value. 1: invert; 0: do not invert. (R/W)

GPIO_FUNC*m*_IN_SEL Selection control for peripheral input *m*. A value of 0-39 selects which of the 40 GPIO Matrix input pins this signal is connected to, or 0x38 for a constantly high input or 0x30 for a constantly low input. (R/W)

Register 4.32: GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-19, 21-23, 25-27, 32-33) (0x530+0x4**n*)

(reserved)	GPIO_FUNC*n*_OEN_INV_SEL	GPIO_FUNC*n*_OEN_SEL	GPIO_FUNC*n*_OUT_INV_SEL	GPIO_FUNC*n*_OUT_SEL	
31 … 12	11	10	9	8 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x	x	x	x x x x x x x x x	Reset

GPIO_FUNC*n*_OEN_INV_SEL 1: Invert the output enable signal; 0: do not invert the output enable signal. (R/W)

GPIO_FUNC*n*_OEN_SEL 1: Force the output enable signal to be sourced from bit *n* of GPIO_ENABLE_REG; 0: use output enable signal from peripheral. (R/W)

GPIO_FUNC*n*_OUT_INV_SEL 1: Invert the output value; 0: do not invert the output value. (R/W)

GPIO_FUNC*n*_OUT_SEL Selection control for GPIO output *n*. A value of *s* (0<=*s*<256) connects peripheral output *s* to GPIO output *n*. A value of 256 selects bit *n* of GPIO_OUT_REG/GPIO_OUT1_REG and GPIO_ENABLE_REG/GPIO_ENABLE1_REG as the output value and output enable. (R/W)

Register 4.33: IO_MUX_PIN_CTRL (0x3FF49000)

(reserved)	PIN_CTRL_CLK3	PIN_CTRL_CLK2	PIN_CTRL_CLK1	
31 … 12	11 … 8	7 … 4	3 … 0	
0x0	0x0	0x0	0x0	Reset

If you want to output clock for I2S0 to:
CLK_OUT1, then set PIN_CTRL[3:0] = 0x0;
CLK_OUT2, then set PIN_CTRL[3:0] = 0x0 and PIN_CTRL[7:4] = 0x0;
CLK_OUT3, then set PIN_CTRL[3:0] = 0x0 and PIN_CTRL[11:8] = 0x0.
If you want to output clock for I2S1 to:
CLK_OUT1, then set PIN_CTRL[3:0] = 0xF;
CLK_OUT2, then set PIN_CTRL[3:0] = 0xF and PIN_CTRL[7:4] = 0x0;
CLK_OUT3, then set PIN_CTRL[3:0] = 0xF and PIN_CTRL[11:8] = 0x0. (R/W)

Note:
Only the above mentioned combinations of clock source and clock output pins are possible.
The CLK_OUT1-3 can be found in the IO_MUX Pad Summary.

Register 4.34: IO_MUX_*x*_REG (*x*: GPIO0-GPIO39) (0x10+4**x*)

(reserved)	IO_*x*_MCU_SEL	IO_*x*_FUNC_DRV	IO_*x*_FUNC_IE	IO_*x*_FUNC_WPU	IO_*x*_FUNC_WPD	IO_*x*_MCU_DRV	IO_*x*_MCU_IE	IO_*x*_MCU_WPU	IO_*x*_MCU_WPD	IO_*x*_SLP_SEL	IO_*x*_MCU_OE	
31 15	14 12	11 10	9	8	7	6 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x2	0	0	0	0x0	0	0	0	0	0	Reset

IO_*x*_MCU_SEL Select the IO_MUX function for this signal. 0 selects Function 1, 1 selects Function 2, etc. (R/W)

IO_*x*_FUNC_DRV Select the drive strength of the pad. A higher value corresponds with a higher strength. (R/W)

IO_*x*_FUNC_IE Input enable of the pad. 1: input enabled; 0: input disabled. (R/W)

IO_*x*_FUNC_WPU Pull-up enable of the pad. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

IO_*x*_FUNC_WPD Pull-down enable of the pad. 1: internal pull-down enabled, 0: internal pull-down disabled. (R/W)

IO_*x*_MCU_DRV Select the drive strength of the pad during sleep mode. A higher value corresponds with a higher strength. (R/W)

IO_*x*_MCU_IE Input enable of the pad during sleep mode. 1: input enabled; 0: input disabled. (R/W)

IO_*x*_MCU_WPU Pull-up enable of the pad during sleep mode. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

IO_*x*_MCU_WPD Pull-down enable of the pad during sleep mode. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

IO_*x*_SLP_SEL Sleep mode selection of this pad. Set to 1 to put the pad in sleep mode. (R/W)

IO_*x*_MCU_OE Output enable of the pad in sleep mode. 1: enable output; 0: disable output. (R/W)

Register 4.35: RTCIO_RTC_GPIO_OUT_REG (0x0000)

RTCIO_RTC_GPIO_OUT_DATA	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_OUT_DATA GPIO0-17 output register. Bit14 is GPIO[0], bit15 is GPIO[1], etc. (R/W)

Register 4.36: RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0004)

RTCIO_RTC_GPIO_OUT_DATA_W1TS	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_OUT_DATA_W1TS GPIO0-17 output set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be set. (WO)

Register 4.37: RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0008)

RTCIO_RTC_GPIO_OUT_DATA_W1TC	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_OUT_DATA_W1TC GPIO0-17 output clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_OUT will be cleared. (WO)

Register 4.38: RTCIO_RTC_GPIO_ENABLE_REG (0x000C)

RTCIO_RTC_GPIO_ENABLE	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_ENABLE GPIO0-17 output enable. Bit14 is GPIO[0], bit15 is GPIO[1], etc. 1 means this GPIO pad is output. (R/W)

Register 4.39: RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0010)

RTCIO_RTC_GPIO_ENABLE_W1TS	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_ENABLE_W1TS GPIO0-17 output enable set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be set. (WO)

Register 4.40: RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0014)

RTCIO_RTC_GPIO_ENABLE_W1TC	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

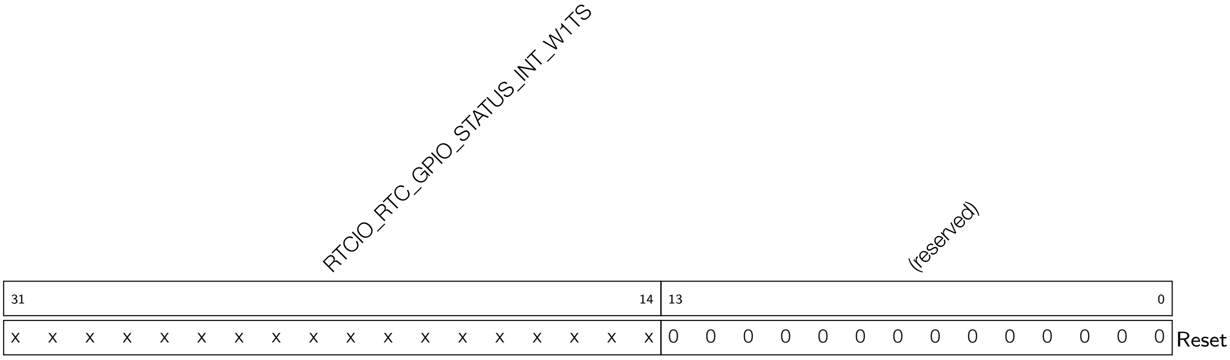
RTCIO_RTC_GPIO_ENABLE_W1TC GPIO0-17 output enable clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_ENABLE will be cleared. (WO)

Register 4.41: RTCIO_RTC_GPIO_STATUS_REG (0x0018)

RTCIO_RTC_GPIO_STATUS_INT	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

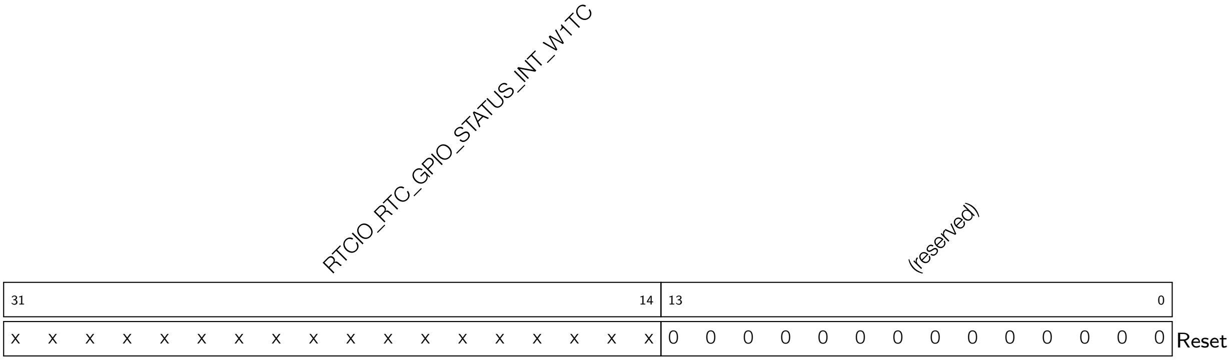
RTCIO_RTC_GPIO_STATUS_INT GPIO0-17 interrupt status. Bit14 is GPIO[0], bit15 is GPIO[1], etc. This register should be used together with RTCIO_RTC_GPIO_PIN*n*_INT_TYPE in RTCIO_RTC_GPIO_PIN*n*_REG. 1: corresponding interrupt; 0: no interrupt. (R/W)

Register 4.42: RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x001C)


RTCIO_RTC_GPIO_STATUS_INT_W1TS
(reserved)
31
14
13
0
x x x x x x x x x x x x x x x x x x
0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset

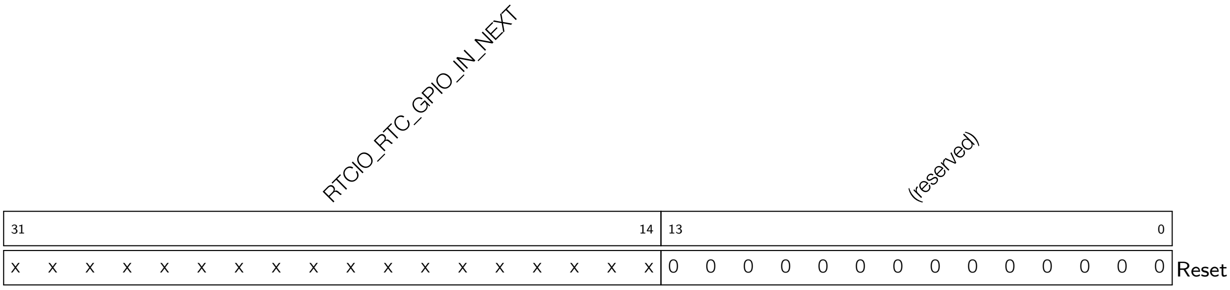
RTCIO_RTC_GPIO_STATUS_INT_W1TS GPIO0-17 interrupt set register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be set. (WO)

Register 4.43: RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0020)


RTCIO_RTC_GPIO_STATUS_INT_W1TC
(reserved)
31
14
13
0
x x x x x x x x x x x x x x x x x x
0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset

RTCIO_RTC_GPIO_STATUS_INT_W1TC GPIO0-17 interrupt clear register. For every bit that is 1 in the value written here, the corresponding bit in RTCIO_RTC_GPIO_STATUS_INT will be cleared. (WO)

Register 4.44: RTCIO_RTC_GPIO_IN_REG (0x0024)



RTCIO_RTC_GPIO_IN_NEXT GPIO0-17 input value. Bit14 is GPIO[0], bit15 is GPIO[1], etc. Each bit represents a pad input value, 1 for high level, and 0 for low level. (RO)

Register 4.45: RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-17) (28+4**n*)

	(reserved)	RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE	RTCIO_RTC_GPIO_PIN*n*_INT_TYPE	(reserved)	RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER	(reserved)
	31 11	10	9 7	6 3	2	1 0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x	x x x	0 0 0 0	x	0 0

RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE GPIO wake-up enable. This will only wake up the ESP32 from Light-sleep. (R/W)

RTCIO_RTC_GPIO_PIN*n*_INT_TYPE GPIO interrupt type selection. (R/W)

0: GPIO interrupt disable;

1: rising edge trigger;

2: falling edge trigger;

3: any edge trigger;

4: low level trigger;

5: high level trigger.

RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER Pad driver selection. 0: normal output; 1: open drain. (R/W)

Register 4.46: RTCIO_DIG_PAD_HOLD_REG (0x0074)

31 0	
0	Reset

RTCIO_DIG_PAD_HOLD_REG Selects the digital pads which should be put on hold. While 0 allows normal operation, 1 puts the pad on hold. (R/W)

Name	Description
Bit[0]	Set to 1 to enable the Hold function of pad U0RTD
Bit[1]	Set to 1 to enable the Hold function of pad U0TXD
Bit[2]	Set to 1 to enable the Hold function of pad SD_CLK
Bit[3]	Set to 1 to enable the Hold function of pad SD_DATA0
Bit[4]	Set to 1 to enable the Hold function of pad SD_DATA1
Bit[5]	Set to 1 to enable the Hold function of pad SD_DATA2
Bit[6]	Set to 1 to enable the Hold function of pad SD_DATA3
Bit[7]	Set to 1 to enable the Hold function of pad SD_CMD
Bit[8]	Set to 1 to enable the Hold function of pad GPIO5
Bit[9]	Set to 1 to enable the Hold function of pad GPIO16
Bit[10]	Set to 1 to enable the Hold function of pad GPIO17
Bit[11]	Set to 1 to enable the Hold function of pad GPIO18
Bit[12]	Set to 1 to enable the Hold function of pad GPIO19
Bit[13]	Set to 1 to enable the Hold function of pad GPIO20
Bit[14]	Set to 1 to enable the Hold function of pad GPIO21
Bit[15]	Set to 1 to enable the Hold function of pad GPIO22
Bit[16]	Set to 1 to enable the Hold function of pad GPIO23

Register 4.47: RTCIO_HALL_SENS_REG (0x0078)

RTCIO_HALL_XPD_HALL	RTCIO_HALL_PHASE	(reserved)	
31	30	29 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_HALL_XPD_HALL Power on hall sensor and connect to VP and VN. (R/W)

RTCIO_HALL_PHASE Reverse the polarity of the hall sensor. (R/W)

Register 4.48: RTCIO_SENSOR_PADS_REG (0x007C)

RTCIO_SENSOR_SENSE1_HOLD	RTCIO_SENSOR_SENSE2_HOLD	RTCIO_SENSOR_SENSE3_HOLD	RTCIO_SENSOR_SENSE4_HOLD	RTCIO_SENSOR_SENSE1_MUX_SEL	RTCIO_SENSOR_SENSE2_MUX_SEL	RTCIO_SENSOR_SENSE3_MUX_SEL	RTCIO_SENSOR_SENSE4_MUX_SEL	RTCIO_SENSOR_SENSE1_FUN_SEL	RTCIO_SENSOR_SENSE1_SLP_SEL	RTCIO_SENSOR_SENSE1_SLP_IE	RTCIO_SENSOR_SENSE1_FUN_IE	RTCIO_SENSOR_SENSE2_FUN_SEL	RTCIO_SENSOR_SENSE2_SLP_SEL	RTCIO_SENSOR_SENSE2_SLP_IE	RTCIO_SENSOR_SENSE2_FUN_IE	RTCIO_SENSOR_SENSE3_FUN_SEL	RTCIO_SENSOR_SENSE3_SLP_SEL	RTCIO_SENSOR_SENSE3_SLP_IE	RTCIO_SENSOR_SENSE3_FUN_IE	RTCIO_SENSOR_SENSE4_FUN_SEL	RTCIO_SENSOR_SENSE4_SLP_SEL	RTCIO_SENSOR_SENSE4_SLP_IE	RTCIO_SENSOR_SENSE4_FUN_IE	(reserved)	
31	30	29	28	27	26	25	24	23 22	21	20	19	18 17	16	15	14	13 12	11	10	9	8 7	6	5	4	3 0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0	Reset

RTCIO_SENSOR_SENSE*n*_HOLD Set to 1 to hold the output value on sense*n*; 0 is for normal operation. (R/W)

RTCIO_SENSOR_SENSE*n*_MUX_SEL 1: route sense*n* to the RTC block; 0: route sense*n* to the digital IO_MUX. (R/W)

RTCIO_SENSOR_SENSE*n*_FUN_SEL Select the RTC IO_MUX function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_SENSOR_SENSE*n*_SLP_SEL Selection of sleep mode for the pad: set to 1 to put the pad in sleep mode. (R/W)

RTCIO_SENSOR_SENSE*n*_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_SENSOR_SENSE*n*_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

Register 4.49: RTCIO_ADC_PAD_REG (0x0080)

RTCIO_ADC_ADC1_HOLD	RTCIO_ADC_ADC2_HOLD	RTCIO_ADC_ADC1_MUX_SEL	RTCIO_ADC_ADC2_MUX_SEL	RTCIO_ADC_ADC1_FUN_SEL	RTCIO_ADC_ADC1_SLP_SEL	RTCIO_ADC_ADC1_SLP_IE	RTCIO_ADC_ADC1_FUN_IE	RTCIO_ADC_ADC2_FUN_SEL	RTCIO_ADC_ADC2_SLP_SEL	RTCIO_ADC_ADC2_SLP_IE	RTCIO_ADC_ADC2_FUN_IE	(reserved)	
31	30	29	28	27 26	25	24	23	22 21	20	19	18	17 0	
0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_ADC_ADC*n*_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_ADC_ADC*n*_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route pad to the RTC block.

RTCIO_ADC_ADC*n*_FUN_SEL Select the RTC function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_ADC_ADC*n*_SLP_SEL Signal selection of pad's sleep mode. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_ADC_ADC*n*_SLP_IE Input enable of the pad in sleep mode. 1 enabled; 0 disabled. (R/W)

RTCIO_ADC_ADC*n*_FUN_IE Input enable of the pad. 1 enabled; 0 disabled. (R/W)

Register 4.50: RTCIO_PAD_DAC1_REG (0x0084)

RTCIO_PAD_PDAC1_DRV	RTCIO_PAD_PDAC1_HOLD	RTCIO_PAD_PDAC1_RDE	RTCIO_PAD_PDAC1_RUE	RTCIO_PAD_PDAC1_DAC	RTCIO_PAD_PDAC1_XPD_DAC	RTCIO_PAD_PDAC1_MUX_SEL	RTCIO_PAD_PDAC1_FUN_SEL	RTCIO_PAD_PDAC1_SLP_SEL	RTCIO_PAD_PDAC1_SLP_IE	RTCIO_PAD_PDAC1_SLP_OE	RTCIO_PAD_PDAC1_FUN_IE	RTCIO_PAD_PDAC1_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	9 0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC1_DRV Select the drive strength of the pad. (R/W)

RTCIO_PAD_PDAC1_HOLD Set to 1 to hold the output value on the pad; set to 0 for normal operation. (R/W)

RTCIO_PAD_PDAC1_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_PAD_PDAC1_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_PAD_PDAC1_DAC PAD DAC1 output value. (R/W)

RTCIO_PAD_PDAC1_XPD_DAC Power on DAC1. Usually, PDAC1 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

RTCIO_PAD_PDAC1_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route to the RTC block.

RTCIO_PAD_PDAC1_FUN_SEL the functional selection signal of the pad. (R/W)

RTCIO_PAD_PDAC1_SLP_SEL Sleep mode selection signal of the pad. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_PAD_PDAC1_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_SLP_OE Output enable of the pad. 1: enabled ; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_FUN_IE Input enable of the pad. 1: enabled it; 0: disabled. (R/W)

RTCIO_PAD_PDAC1_DAC_XPD_FORCE Power on DAC1. Usually, we need to tristate PDAC1 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

Register 4.51: RTCIO_PAD_DAC2_REG (0x0088)

RTCIO_PAD_PDAC2_DRV	RTCIO_PAD_PDAC2_HOLD	RTCIO_PAD_PDAC2_RDE	RTCIO_PAD_PDAC2_RUE	RTCIO_PAD_PDAC2_DAC	RTCIO_PAD_PDAC2_XPD_DAC	RTCIO_PAD_PDAC2_MUX_SEL	RTCIO_PAD_PDAC2_FUN_SEL	RTCIO_PAD_PDAC2_SLP_SEL	RTCIO_PAD_PDAC2_SLP_IE	RTCIO_PAD_PDAC2_SLP_OE	RTCIO_PAD_PDAC2_FUN_IE	RTCIO_PAD_PDAC2_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	9 0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC2_DRV Select the drive strength of the pad. (R/W)

RTCIO_PAD_PDAC2_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_PAD_PDAC2_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_PAD_PDAC2_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_PAD_PDAC2_DAC PAD DAC2 output value. (R/W)

RTCIO_PAD_PDAC2_XPD_DAC Power on DAC2. PDAC2 needs to be tristated if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

RTCIO_PAD_PDAC2_MUX_SEL 0: route pad to the digital IO_MUX; (R/W)
1: route to the RTC block.

RTCIO_PAD_PDAC2_FUN_SEL Select the RTC function for this pad. 0: select Function 0; 1: select Function 1. (R/W)

RTCIO_PAD_PDAC2_SLP_SEL Sleep mode selection signal of the pad. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_PAD_PDAC2_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_SLP_OE Output enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_PAD_PDAC2_DAC_XPD_FORCE Power on DAC2. Usually, we need to tristate PDAC2 if we power on the DAC, i.e. IE=0, OE=0, RDE=0, RUE=0. (R/W)

Register 4.52: RTCIO_XTAL_32K_PAD_REG (0x008C)

RTCIO_XTAL_X32N_DRV	RTCIO_XTAL_X32N_HOLD	RTCIO_XTAL_X32N_RDE	RTCIO_XTAL_X32N_RUE	RTCIO_XTAL_X32P_DRV	RTCIO_XTAL_X32P_HOLD	RTCIO_XTAL_X32P_RDE	RTCIO_XTAL_X32P_RUE	RTCIO_XTAL_DAC_XTAL_32K	RTCIO_XTAL_XPD_XTAL_32K	RTCIO_XTAL_X32N_MUX_SEL	RTCIO_XTAL_X32P_MUX_SEL	RTCIO_XTAL_X32N_FUN_SEL	RTCIO_XTAL_X32N_SLP_SEL	RTCIO_XTAL_X32N_SLP_IE	RTCIO_XTAL_X32N_SLP_OE	RTCIO_XTAL_X32N_FUN_IE	RTCIO_XTAL_X32P_FUN_SEL	RTCIO_XTAL_X32P_SLP_SEL	RTCIO_XTAL_X32P_SLP_IE	RTCIO_XTAL_X32P_SLP_OE	RTCIO_XTAL_X32P_FUN_IE	RTCIO_XTAL_DRES_XTAL_32K	RTCIO_XTAL_DBIAS_XTAL_32K	(reserved)	
31 30	29	28	27	26 25	24	23	22	21 20	19	18	17	16 15	14	13	12	11	10 9	8	7	6	5	4 3	2 1	0	
2	0	0	0	2	0	0	0	0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1 0	0 0	0	Reset

RTCIO_XTAL_X32N_DRV Select the drive strength of the pad. (R/W)

RTCIO_XTAL_X32N_HOLD Set to 1 to hold the output value on the pad; 0 is for normal operation. (R/W)

RTCIO_XTAL_X32N_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_XTAL_X32N_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_XTAL_X32P_DRV Select the drive strength of the pad. (R/W)

RTCIO_XTAL_X32P_HOLD Set to 1 to hold the output value on the pad, 0 is for normal operation. (R/W)

RTCIO_XTAL_X32P_RDE 1: Pull-down on pad enabled; 0: Pull-down disabled. (R/W)

RTCIO_XTAL_X32P_RUE 1: Pull-up on pad enabled; 0: Pull-up disabled. (R/W)

RTCIO_XTAL_DAC_XTAL_32K 32K XTAL bias current DAC value. (R/W)

RTCIO_XTAL_XPD_XTAL_32K Power up 32 KHz crystal oscillator. (R/W)

RTCIO_XTAL_X32N_MUX_SEL 0: route X32N pad to the digital IO_MUX; 1: route to RTC block. (R/W)

RTCIO_XTAL_X32P_MUX_SEL 0: route X32P pad to the digital IO_MUX; 1: route to RTC block. (R/W)

RTCIO_XTAL_X32N_FUN_SEL Select the RTC function. 0: select function 0; 1: select function 1. (R/W)

RTCIO_XTAL_X32N_SLP_SEL Sleep mode selection. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_XTAL_X32N_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32N_SLP_OE Output enable of the pad. 1: enabled; 0; disabled. (R/W)

RTCIO_XTAL_X32N_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_FUN_SEL Select the RTC function. 0: select function 0; 1: select function 1. (R/W)

RTCIO_XTAL_X32P_SLP_SEL Sleep mode selection. Set this bit to 1 to put the pad to sleep. (R/W)

RTCIO_XTAL_X32P_SLP_IE Input enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_SLP_OE Output enable of the pad in sleep mode. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_X32P_FUN_IE Input enable of the pad. 1: enabled; 0: disabled. (R/W)

RTCIO_XTAL_DRES_XTAL_32K 32K XTAL resistor bias control. (R/W)

RTCIO_XTAL_DBIAS_XTAL_32K 32K XTAL self-bias reference control. (R/W)

Register 4.53: RTCIO_TOUCH_CFG_REG (0x0090)

RTCIO_TOUCH_XPD_BIAS	RTCIO_TOUCH_DREFH	RTCIO_TOUCH_DREFL	RTCIO_TOUCH_DRANGE	RTCIO_TOUCH_DCUR	(reserved)	
31	30 29	28 27	26 25	24 23	22 0	
0	1 1	0 0	1 1	0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_XPD_BIAS Touch sensor bias power on bit. 1: power on; 0: disabled. (R/W)

RTCIO_TOUCH_DREFH Touch sensor saw wave top voltage. (R/W)

RTCIO_TOUCH_DREFL Touch sensor saw wave bottom voltage. (R/W)

RTCIO_TOUCH_DRANGE Touch sensor saw wave voltage range. (R/W)

RTCIO_TOUCH_DCUR Touch sensor bias current. When BIAS_SLEEP is enabled, this setting is available. (R/W)

Register 4.54: RTCIO_TOUCH_PAD*n*_REG (*n*: 0-9) (94+4**n*)

(reserved)	RTCIO_TOUCH_PAD*n*_DAC	RTCIO_TOUCH_PAD*n*_START	RTCIO_TOUCH_PAD*n*_TIE_OPT	RTCIO_TOUCH_PAD*n*_XPD	RTCIO_TOUCH_PAD*n*_TO_GPIO	(reserved)	
31 26	25 23	22	21	20	19	18 0	
0 0 0 0 0 0	0x4	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_PAD*n*_DAC Touch sensor slope control. 3-bit for each touch pad, defaults to 100. (R/W)

RTCIO_TOUCH_PAD*n*_START Start touch sensor. (R/W)

RTCIO_TOUCH_PAD*n*_TIE_OPT Default touch sensor tie option. 0: tie low; 1: tie high. (R/W)

RTCIO_TOUCH_PAD*n*_XPD Touch sensor power on. (R/W)

RTCIO_TOUCH_PAD*n*_TO_GPIO Connect the RTC pad input to digital pad input; 0 is available. (R/W)

Register 4.55: RTCIO_EXT_WAKEUP0_REG (0x00BC)

RTCIO_EXT_WAKEUP0_SEL	(reserved)	
31 27	26 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_EXT_WAKEUP0_SEL GPIO[0-17] can be used to wake up the chip when the chip is in the sleep mode. This register prompts the pad source to wake up the chip when the latter is in deep/light sleep mode. 0: select GPIO0; 1: select GPIO2, etc. (R/W)

Register 4.56: RTCIO_XTL_EXT_CTR_REG (0x00C0)

RTCIO_XTL_EXT_CTR_SEL	(reserved)	
31 27	26 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_XTL_EXT_CTR_SEL Select the external crystal power down enable source to get into sleep mode. 0: select GPIO0; 1: select GPIO2, etc. The input value on this pin XOR RTCIO_RTC_EXT_XTAL_CONF_REG[30] is the crystal power down enable signal. (R/W)

Register 4.57: RTCIO_SAR_I2C_IO_REG (0x00C4)

RTCIO_SAR_I2C_SDA_SEL	RTCIO_SAR_I2C_SCL_SEL	(reserved)	
31 30	29 28	27 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_SAR_I2C_SDA_SEL Selects a different pad as the RTC I2C SDA signal. 0: use pad TOUCH_PAD[1]; 1: use pad TOUCH_PAD[3]. (R/W)

RTCIO_SAR_I2C_SCL_SEL Selects a different pad as the RTC I2C SCL signal. 0: use pad TOUCH_PAD[0]; 1: use pad TOUCH_PAD[2]. (R/W)

5. DPort Register

5.1 Introduction

The ESP32 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. The DPort registers control clock management (clock gating), power management, and the configuration of peripherals and core-system modules. The system arranges each module with configuration registers contained in the DPort Register.

5.2 Features

DPort registers correspond to different peripheral blocks and core modules:

- System and memory
- Reset and clock
- Interrupt matrix
- DMA
- PID/MPU/MMU
- APP_CPU
- Peripheral clock gating and reset

5.3 Functional Description

5.3.1 System and Memory Register

The following registers are used for system and memory configuration, such as cache configuration and memory remapping. For a detailed description of these registers, please refer to Chapter System and Memory.

- DPORT_PRO_BOOT_REMAP_CTRL_REG
- DPORT_APP_BOOT_REMAP_CTRL_REG
- DPORT_CACHE_MUX_MODE_REG

5.3.2 Reset and Clock Registers

The following register is used for Reset and Clock. For a detailed description of the register, please refer to Reset and Clock.

- DPORT_CPU_PER_CONF_REG

5.3.3 Interrupt Matrix Register

The following registers are used for configuring and mapping interrupts through the interrupt matrix. For a detailed description of the registers, please refer to Interrupt Matrix.

- DPORT_CPU_INTR_FROM_CPU_0_REG
- DPORT_CPU_INTR_FROM_CPU_1_REG
- DPORT_CPU_INTR_FROM_CPU_2_REG
- DPORT_CPU_INTR_FROM_CPU_3_REG
- DPORT_PRO_INTR_STATUS_0_REG
- DPORT_PRO_INTR_STATUS_1_REG
- DPORT_PRO_INTR_STATUS_2_REG
- DPORT_APP_INTR_STATUS_0_REG
- DPORT_APP_INTR_STATUS_1_REG
- DPORT_APP_INTR_STATUS_2_REG
- DPORT_PRO_MAC_INTR_MAP_REG
- DPORT_PRO_MAC_NMI_MAP_REG
- DPORT_PRO_BB_INT_MAP_REG
- DPORT_PRO_BT_MAC_INT_MAP_REG
- DPORT_PRO_BT_BB_INT_MAP_REG
- DPORT_PRO_BT_BB_NMI_MAP_REG
- DPORT_PRO_RWBT_IRQ_MAP_REG
- DPORT_PRO_RWBLE_IRQ_MAP_REG
- DPORT_PRO_RWBT_NMI_MAP_REG
- DPORT_PRO_RWBLE_NMI_MAP_REG
- DPORT_PRO_SLC0_INTR_MAP_REG
- DPORT_PRO_SLC1_INTR_MAP_REG
- DPORT_PRO_UHCI0_INTR_MAP_REG
- DPORT_PRO_UHCI1_INTR_MAP_REG
- DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG
- DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG
- DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG
- DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG
- DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG
- DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG
- DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG

- DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG
- DPORT_PRO_GPIO_INTERRUPT_MAP_REG
- DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG
- DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG
- DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG
- DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG
- DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG
- DPORT_PRO_SPI_INTR_0_MAP_REG
- DPORT_PRO_SPI_INTR_1_MAP_REG
- DPORT_PRO_SPI_INTR_2_MAP_REG
- DPORT_PRO_SPI_INTR_3_MAP_REG
- DPORT_PRO_I2S0_INT_MAP_REG
- DPORT_PRO_I2S1_INT_MAP_REG
- DPORT_PRO_UART_INTR_MAP_REG
- DPORT_PRO_UART1_INTR_MAP_REG
- DPORT_PRO_UART2_INTR_MAP_REG
- DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG
- DPORT_PRO_EMAC_INT_MAP_REG
- DPORT_PRO_PWM0_INTR_MAP_REG
- DPORT_PRO_PWM1_INTR_MAP_REG
- DPORT_PRO_PWM2_INTR_MAP_REG
- DPORT_PRO_PWM3_INTR_MAP_REG
- DPORT_PRO_LEDC_INT_MAP_REG
- DPORT_PRO_EFUSE_INT_MAP_REG
- DPORT_PRO_CAN_INT_MAP_REG
- DPORT_PRO_RTC_CORE_INTR_MAP_REG
- DPORT_PRO_RMT_INTR_MAP_REG
- DPORT_PRO_PCNT_INTR_MAP_REG
- DPORT_PRO_I2C_EXT0_INTR_MAP_REG
- DPORT_PRO_I2C_EXT1_INTR_MAP_REG
- DPORT_PRO_RSA_INTR_MAP_REG
- DPORT_PRO_SPI1_DMA_INT_MAP_REG
- DPORT_PRO_SPI2_DMA_INT_MAP_REG
- DPORT_PRO_SPI3_DMA_INT_MAP_REG

- DPORT_PRO_WDG_INT_MAP_REG
- DPORT_PRO_TIMER_INT1_MAP_REG
- DPORT_PRO_TIMER_INT2_MAP_REG
- DPORT_PRO_TG_T0_EDGE_INT_MAP_REG
- DPORT_PRO_TG_T1_EDGE_INT_MAP_REG
- DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG
- DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG
- DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG
- DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG
- DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG
- DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG
- DPORT_PRO_MMU_IA_INT_MAP_REG
- DPORT_PRO_MPU_IA_INT_MAP_REG
- DPORT_PRO_CACHE_IA_INT_MAP_REG
- DPORT_APP_MAC_INTR_MAP_REG
- DPORT_APP_MAC_NMI_MAP_REG
- DPORT_APP_BB_INT_MAP_REG
- DPORT_APP_BT_MAC_INT_MAP_REG
- DPORT_APP_BT_BB_INT_MAP_REG
- DPORT_APP_BT_BB_NMI_MAP_REG
- DPORT_APP_RWBT_IRQ_MAP_REG
- DPORT_APP_RWBLE_IRQ_MAP_REG
- DPORT_APP_RWBT_NMI_MAP_REG
- DPORT_APP_RWBLE_NMI_MAP_REG
- DPORT_APP_SLC0_INTR_MAP_REG
- DPORT_APP_SLC1_INTR_MAP_REG
- DPORT_APP_UHCI0_INTR_MAP_REG
- DPORT_APP_UHCI1_INTR_MAP_REG
- DPORT_APP_TG_T0_LEVEL_INT_MAP_REG
- DPORT_APP_TG_T1_LEVEL_INT_MAP_REG
- DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG
- DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG
- DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG
- DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG

- DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG
- DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG
- DPORT_APP_GPIO_INTERRUPT_MAP_REG
- DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG
- DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG
- DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG
- DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG
- DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG
- DPORT_APP_SPI_INTR_0_MAP_REG
- DPORT_APP_SPI_INTR_1_MAP_REG
- DPORT_APP_SPI_INTR_2_MAP_REG
- DPORT_APP_SPI_INTR_3_MAP_REG
- DPORT_APP_I2S0_INT_MAP_REG
- DPORT_APP_I2S1_INT_MAP_REG
- DPORT_APP_UART_INTR_MAP_REG
- DPORT_APP_UART1_INTR_MAP_REG
- DPORT_APP_UART2_INTR_MAP_REG
- DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG
- DPORT_APP_EMAC_INT_MAP_REG
- DPORT_APP_PWM0_INTR_MAP_REG
- DPORT_APP_PWM1_INTR_MAP_REG
- DPORT_APP_PWM2_INTR_MAP_REG
- DPORT_APP_PWM3_INTR_MAP_REG
- DPORT_APP_LEDC_INT_MAP_REG
- DPORT_APP_EFUSE_INT_MAP_REG
- DPORT_APP_CAN_INT_MAP_REG
- DPORT_APP_RTC_CORE_INTR_MAP_REG
- DPORT_APP_RMT_INTR_MAP_REG
- DPORT_APP_PCNT_INTR_MAP_REG
- DPORT_APP_I2C_EXT0_INTR_MAP_REG
- DPORT_APP_I2C_EXT1_INTR_MAP_REG
- DPORT_APP_RSA_INTR_MAP_REG
- DPORT_APP_SPI1_DMA_INT_MAP_REG
- DPORT_APP_SPI2_DMA_INT_MAP_REG

- DPORT_APP_SPI3_DMA_INT_MAP_REG
- DPORT_APP_WDG_INT_MAP_REG
- DPORT_APP_TIMER_INT1_MAP_REG
- DPORT_APP_TIMER_INT2_MAP_REG
- DPORT_APP_TG_T0_EDGE_INT_MAP_REG
- DPORT_APP_TG_T1_EDGE_INT_MAP_REG
- DPORT_APP_TG_WDT_EDGE_INT_MAP_REG
- DPORT_APP_TG_LACT_EDGE_INT_MAP_REG
- DPORT_APP_TG1_T0_EDGE_INT_MAP_REG
- DPORT_APP_TG1_T1_EDGE_INT_MAP_REG
- DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG
- DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG
- DPORT_APP_MMU_IA_INT_MAP_REG
- DPORT_APP_MPU_IA_INT_MAP_REG
- DPORT_APP_CACHE_IA_INT_MAP_REG

5.3.4 DMA Registers

The following register is used for the SPI DMA configuration. For a detailed description of the register, please refer to DMA.

- DPORT_SPI_DMA_CHAN_SEL_REG

5.3.5 PID/MPU/MMU Registers

The following registers are used for PID/MPU/MMU configuration and operation control. For a detailed description of the registers, please refer to PID/MPU/MMU.

- DPORT_PRO_CACHE_CTRL_REG
- DPORT_APP_CACHE_CTRL_REG
- DPORT_IMMU_PAGE_MODE_REG
- DPORT_DMMU_PAGE_MODE_REG
- DPORT_AHB_MPU_TABLE_0_REG
- DPORT_AHB_MPU_TABLE_1_REG
- DPORT_AHBLITE_MPU_TABLE_UART_REG
- DPORT_AHBLITE_MPU_TABLE_SPI1_REG
- DPORT_AHBLITE_MPU_TABLE_SPI0_REG
- DPORT_AHBLITE_MPU_TABLE_GPIO_REG
- DPORT_AHBLITE_MPU_TABLE_FE2_REG

- DPORT_AHBLITE_MPU_TABLE_FE_REG
- DPORT_AHBLITE_MPU_TABLE_TIMER_REG
- DPORT_AHBLITE_MPU_TABLE_RTC_REG
- DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG
- DPORT_AHBLITE_MPU_TABLE_WDG_REG
- DPORT_AHBLITE_MPU_TABLE_HINF_REG
- DPORT_AHBLITE_MPU_TABLE_UHCI1_REG
- DPORT_AHBLITE_MPU_TABLE_I2S0_REG
- DPORT_AHBLITE_MPU_TABLE_UART1_REG
- DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG
- DPORT_AHBLITE_MPU_TABLE_UHCI0_REG
- DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG
- DPORT_AHBLITE_MPU_TABLE_RMT_REG
- DPORT_AHBLITE_MPU_TABLE_PCNT_REG
- DPORT_AHBLITE_MPU_TABLE_SLC_REG
- DPORT_AHBLITE_MPU_TABLE_LEDC_REG
- DPORT_AHBLITE_MPU_TABLE_EFUSE_REG
- DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG
- DPORT_AHBLITE_MPU_TABLE_PWM0_REG
- DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG
- DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG
- DPORT_AHBLITE_MPU_TABLE_SPI2_REG
- DPORT_AHBLITE_MPU_TABLE_SPI3_REG
- DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG
- DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG
- DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG
- DPORT_AHBLITE_MPU_TABLE_EMAC_REG
- DPORT_AHBLITE_MPU_TABLE_PWM1_REG
- DPORT_AHBLITE_MPU_TABLE_I2S1_REG
- DPORT_AHBLITE_MPU_TABLE_UART2_REG
- DPORT_AHBLITE_MPU_TABLE_PWM2_REG
- DPORT_AHBLITE_MPU_TABLE_PWM3_REG
- DPORT_AHBLITE_MPU_TABLE_PWR_REG
- DPORT_IMMU_TABLE0_REG

- DPORT_IMMU_TABLE1_REG
- DPORT_IMMU_TABLE2_REG
- DPORT_IMMU_TABLE3_REG
- DPORT_IMMU_TABLE4_REG
- DPORT_IMMU_TABLE5_REG
- DPORT_IMMU_TABLE6_REG
- DPORT_IMMU_TABLE7_REG
- DPORT_IMMU_TABLE8_REG
- DPORT_IMMU_TABLE9_REG
- DPORT_IMMU_TABLE10_REG
- DPORT_IMMU_TABLE11_REG
- DPORT_IMMU_TABLE12_REG
- DPORT_IMMU_TABLE13_REG
- DPORT_IMMU_TABLE14_REG
- DPORT_IMMU_TABLE15_REG
- DPORT_DMMU_TABLE0_REG
- DPORT_DMMU_TABLE1_REG
- DPORT_DMMU_TABLE2_REG
- DPORT_DMMU_TABLE3_REG
- DPORT_DMMU_TABLE4_REG
- DPORT_DMMU_TABLE5_REG
- DPORT_DMMU_TABLE6_REG
- DPORT_DMMU_TABLE7_REG
- DPORT_DMMU_TABLE8_REG
- DPORT_DMMU_TABLE9_REG
- DPORT_DMMU_TABLE10_REG
- DPORT_DMMU_TABLE11_REG
- DPORT_DMMU_TABLE12_REG
- DPORT_DMMU_TABLE13_REG
- DPORT_DMMU_TABLE14_REG
- DPORT_DMMU_TABLE15_REG

5.3.6 APP_CPU Controller Registers

DPort registers are used for some basic configuration of the APP_CPU, such as performing a stalling execution, and for configuring the ROM boot jump address.

- APP_CPU is reset when DPORT_APPCPU_RESETTING=1. It is released when DPORT_APPCPU_RESETTING=0.
- When DPORT_APPCPU_CLKGATE_EN=0, the APP_CPU clock can be disabled to reduce power consumption.
- When DPORT_APPCPU_RUNSTALL=1, the APP_CPU can be put into a stalled state.
- When APP_CPU is booted up with a ROM code, it will jump to the address stored in the DPORT_APPCPU_BOOT_ADDR register.

5.3.7 Peripheral Clock Gating and Reset

Reset and clock gating registers covered in this section are active-high registers. Note that the reset bits are not self-cleared by hardware. When a clock-gating register bit is set to 1, the corresponding clock is enabled. Setting the register bit to 0 disables the clock. Setting a reset register bit to 1 puts the peripheral in a reset state, while setting the register bit to 0 disables the reset state, thus enabling normal operation.

- DPORT_PERI_CLK_EN_REG: enables the hardware accelerator clock.
  - BIT4, Digital Signature
  - BIT3, Secure boot
  - BIT2, RSA Accelerator
  - BIT1, SHA Accelerator
  - BIT0, AES Accelerator
- DPORT_PERI_RST_EN_REG: resets the accelerator.
  - BIT4, Digital Signature
    AES Accelerator and RSA Accelerator will also be reset.
  - BIT3, Secure boot
    AES Accelerator and SHA Accelerator will also be reset.
  - BIT2, RSA Accelerator
  - BIT1, SHA Accelerator
  - BIT0, AES Accelerator
- DPORT_PERIP_CLK_EN_REG=1: enables the peripheral clock.
  - BIT26, PWM3
  - BIT25, PWM2
  - BIT24, UART MEM
    All UART-shared memory. As long as a UART is working, the UART memory clock cannot be in the gating state.
  - BIT23, UART2

  - BIT22, SPI_DMA
  - BIT21, I2S1
  - BIT20, PWM1
  - BIT19, CAN
  - BIT18, I2C1
  - BIT17, PWM0
  - BIT16, SPI3
  - BIT15, Timer Group1
  - BIT14, eFuse
  - BIT13, Timer Group0
  - BIT12, UHCI1
  - BIT11, LED_PWM
  - BIT10, PULSE_CNT
  - BIT9, Remote Controller
  - BIT8, UHCI0
  - BIT7, I2C0
  - BIT6, SPI2
  - BIT5, UART1
  - BIT4, I2S0
  - BIT3, WDG
  - BIT2, UART
  - BIT1, SPI
  - BIT0, Timers

- DPORT_PERIP_RST_EN_REG: resets peripherals
  - BIT26, PWM3
  - BIT25, PWM2
  - BIT24, UART MEM
  - BIT23, UART2
  - BIT22, SPI_DMA
  - BIT21, I2S1
  - BIT20, PWM1
  - BIT19, CAN
  - BIT18, I2C1
  - BIT17, PWM0

  - BIT16, SPI3
  - BIT15, Timer Group1
  - BIT14, eFuse
  - BIT13, Timer Group0
  - BIT12, UHCI1
  - BIT11, LED_PWM
  - BIT10, PULSE_CNT
  - BIT9, Remote Controller
  - BIT8, UHCI0
  - BIT7, I2C0
  - BIT6, SPI2
  - BIT5, UART1
  - BIT4, I2S0
  - BIT3, WDG
  - BIT2, UART
  - BIT1, SPI
  - BIT0, Timers

- DPORT_WIFI_CLK_EN_REG: used for Wi-Fi and BT clock gating.
- DPORT_WIFI_RST_EN_REG: used for Wi-Fi and BT reset.

5.4 Register Summary

Name	Description	Address	Access
PRO_BOOT_REMAP_CTRL_REG	remap mode for PRO_CPU	0x3FF00000	R/W
APP_BOOT_REMAP_CTRL_REG	remap mode for APP_CPU	0x3FF00004	R/W
PERI_CLK_EN_REG	clock gate for peripherals	0x3FF0001C	R/W
PERI_RST_EN_REG	reset for peripherals	0x3FF00020	R/W
APPCPU_CTRL_REG_A_REG	reset for APP_CPU	0x3FF0002C	R/W
APPCPU_CTRL_REG_B_REG	clock gate for APP_CPU	0x3FF00030	R/W
APPCPU_CTRL_REG_C_REG	stall for APP_CPU	0x3FF00034	R/W
APPCPU_CTRL_REG_D_REG	boot address for APP_CPU	0x3FF00038	R/W
PRO_CACHE_CTRL_REG	determines the virtual address mode of the external SRAM	0x3FF00040	R/W
APP_CACHE_CTRL_REG	determines the virtual address mode of the external SRAM	0x3FF00058	R/W
CACHE_MUX_MODE_REG	the mode of the two caches sharing the memory	0x3FF0007C	R/W
IMMU_PAGE_MODE_REG	page size in the MMU for the internal SRAM 0	0x3FF00080	R/W
DMMU_PAGE_MODE_REG	page size in the MMU for the internal SRAM 2	0x3FF00084	R/W
SRAM_PD_CTRL_REG_0_REG	powers down internal SRAM_REG	0x3FF00098	R/W
SRAM_PD_CTRL_REG_1_REG	powers down internal SRAM_REG	0x3FF0009C	R/W
AHB_MPU_TABLE_0_REG	MPU for configuring DMA	0x3FF000B4	R/W
AHB_MPU_TABLE_1_REG	MPU for configuring DMA	0x3FF000B8	R/W
PERIP_CLK_EN_REG	clock gate for peripherals	0x3FF000C0	R/W
PERIP_RST_EN_REG	reset for peripherals	0x3FF000C4	R/W
SLAVE_SPI_CONFIG_REG	enables decryption in external flash	0x3FF000C8	R/W
WIFI_CLK_EN_REG	clock gate for Wi-Fi	0x3FF000CC	R/W
WIFI_RST_EN_REG	reset for Wi-Fi	0x3FF000D0	R/W
CPU_INTR_FROM_CPU_0_REG	interrupt 0 in both CPUs	0x3FF000DC	R/W
CPU_INTR_FROM_CPU_1_REG	interrupt 1 in both CPUs	0x3FF000E0	R/W
CPU_INTR_FROM_CPU_2_REG	interrupt 2 in both CPUs	0x3FF000E4	R/W
CPU_INTR_FROM_CPU_3_REG	interrupt 3 in both CPUs	0x3FF000E8	R/W
PRO_INTR_STATUS_REG_0_REG	PRO_CPU interrupt status 0	0x3FF000EC	RO
PRO_INTR_STATUS_REG_1_REG	PRO_CPU interrupt status 1	0x3FF000F0	RO
PRO_INTR_STATUS_REG_2_REG	PRO_CPU interrupt status 2	0x3FF000F4	RO
APP_INTR_STATUS_REG_0_REG	APP_CPU interrupt status 0	0x3FF000F8	RO
APP_INTR_STATUS_REG_1_REG	APP_CPU interrupt status 1	0x3FF000FC	RO
APP_INTR_STATUS_REG_2_REG	APP_CPU interrupt status 2	0x3FF00100	RO
PRO_MAC_INTR_MAP_REG	interrupt map	0x3FF00104	R/W
PRO_MAC_NMI_MAP_REG	interrupt map	0x3FF00108	R/W
PRO_BB_INT_MAP_REG	interrupt map	0x3FF0010C	R/W
PRO_BT_MAC_INT_MAP_REG	interrupt map	0x3FF00110	R/W
PRO_BT_BB_INT_MAP_REG	interrupt map	0x3FF00114	R/W

Name	Description	Address	Access
PRO_BT_BB_NMI_MAP_REG	interrupt map	0x3FF00118	R/W
PRO_RWBT_IRQ_MAP_REG	interrupt map	0x3FF0011C	R/W
PRO_RWBLE_IRQ_MAP_REG	interrupt map	0x3FF00120	R/W
PRO_RWBT_NMI_MAP_REG	interrupt map	0x3FF00124	R/W
PRO_RWBLE_NMI_MAP_REG	interrupt map	0x3FF00128	R/W
PRO_SLC0_INTR_MAP_REG	interrupt map	0x3FF0012C	R/W
PRO_SLC1_INTR_MAP_REG	interrupt map	0x3FF00130	R/W
PRO_UHCI0_INTR_MAP_REG	interrupt map	0x3FF00134	R/W
PRO_UHCI1_INTR_MAP_REG	interrupt map	0x3FF00138	R/W
PRO_TG_T0_LEVEL_INT_MAP_REG	interrupt map	0x3FF0013C	R/W
PRO_TG_T1_LEVEL_INT_MAP_REG	interrupt map	0x3FF00140	R/W
PRO_TG_WDT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00144	R/W
PRO_TG_LACT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00148	R/W
PRO_TG1_T0_LEVEL_INT_MAP_REG	interrupt map	0x3FF0014C	R/W
PRO_TG1_T1_LEVEL_INT_MAP_REG	interrupt map	0x3FF00150	R/W
PRO_TG1_WDT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00154	R/W
PRO_TG1_LACT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00158	R/W
PRO_GPIO_INTERRUPT_MAP_REG	interrupt map	0x3FF0015C	R/W
PRO_GPIO_INTERRUPT_NMI_MAP_REG	interrupt map	0x3FF00160	R/W
PRO_CPU_INTR_FROM_CPU_0_MAP_REG	interrupt map	0x3FF00164	R/W
PRO_CPU_INTR_FROM_CPU_1_MAP_REG	interrupt map	0x3FF00168	R/W
PRO_CPU_INTR_FROM_CPU_2_MAP_REG	Interrupt map	0x3FF0016C	R/W
PRO_CPU_INTR_FROM_CPU_3_MAP_REG	interrupt map	0x3FF00170	R/W
PRO_SPI_INTR_0_MAP_REG	interrupt map	0x3FF00174	R/W
PRO_SPI_INTR_1_MAP_REG	interrupt map	0x3FF00178	R/W
PRO_SPI_INTR_2_MAP_REG	interrupt map	0x3FF0017C	R/W
PRO_SPI_INTR_3_MAP_REG	interrupt map	0x3FF00180	R/W
PRO_I2S0_INT_MAP_REG	interrupt map	0x3FF00184	R/W
PRO_I2S1_INT_MAP_REG	interrupt map	0x3FF00188	R/W
PRO_UART_INTR_MAP_REG	interrupt map	0x3FF0018C	R/W
PRO_UART1_INTR_MAP_REG	interrupt map	0x3FF00190	R/W
PRO_UART2_INTR_MAP_REG	interrupt map	0x3FF00194	R/W
PRO_SDIO_HOST_INTERRUPT_MAP_REG	interrupt map	0x3FF00198	R/W
PRO_EMAC_INT_MAP_REG	interrupt map	0x3FF0019C	R/W
PRO_PWM0_INTR_MAP_REG	interrupt map	0x3FF001A0	R/W
PRO_PWM1_INTR_MAP_REG	interrupt map	0x3FF001A4	R/W
PRO_PWM2_INTR_MAP_REG	interrupt map	0x3FF001A8	R/W
PRO_PWM3_INTR_MAP_REG	interrupt map	0x3FF001AC	R/W
PRO_LEDC_INT_MAP_REG	interrupt map	0x3FF001B0	R/W
PRO_EFUSE_INT_MAP_REG	interrupt map	0x3FF001B4	R/W
PRO_CAN_INT_MAP_REG	interrupt map	0x3FF001B8	R/W
PRO_RTC_CORE_INTR_MAP_REG	interrupt map	0x3FF001BC	R/W
PRO_RMT_INTR_MAP_REG	interrupt map	0x3FF001C0	R/W
PRO_PCNT_INTR_MAP_REG	interrupt map	0x3FF001C4	R/W

Name	Description	Address	Access
PRO_I2C_EXT0_INTR_MAP_REG	interrupt map	0x3FF001C8	R/W
PRO_I2C_EXT1_INTR_MAP_REG	interrupt map	0x3FF001CC	R/W
PRO_RSA_INTR_MAP_REG	interrupt map	0x3FF001D0	R/W
PRO_SPI1_DMA_INT_MAP_REG	interrupt map	0x3FF001D4	R/W
PRO_SPI2_DMA_INT_MAP_REG	interrupt map	0x3FF001D8	R/W
PRO_SPI3_DMA_INT_MAP_REG	interrupt map	0x3FF001DC	R/W
PRO_WDG_INT_MAP_REG	interrupt map	0x3FF001E0	R/W
PRO_TIMER_INT1_MAP_REG	interrupt map	0x3FF001E4	R/W
PRO_TIMER_INT2_MAP_REG	interrupt map	0x3FF001E8	R/W
PRO_TG_T0_EDGE_INT_MAP_REG	interrupt map	0x3FF001EC	R/W
PRO_TG_T1_EDGE_INT_MAP_REG	interrupt map	0x3FF001F0	R/W
PRO_TG_WDT_EDGE_INT_MAP_REG	interrupt map	0x3FF001F4	R/W
PRO_TG_LACT_EDGE_INT_MAP_REG	interrupt map	0x3FF001F8	R/W
PRO_TG1_T0_EDGE_INT_MAP_REG	interrupt map	0x3FF001FC	R/W
PRO_TG1_T1_EDGE_INT_MAP_REG	interrupt map	0x3FF00200	R/W
PRO_TG1_WDT_EDGE_INT_MAP_REG	interrupt map	0x3FF00204	R/W
PRO_TG1_LACT_EDGE_INT_MAP_REG	interrupt map	0x3FF00208	R/W
PRO_MMU_IA_INT_MAP_REG	interrupt map	0x3FF0020C	R/W
PRO_MPU_IA_INT_MAP_REG	interrupt map	0x3FF00210	R/W
PRO_CACHE_IA_INT_MAP_REG	interrupt map	0x3FF00214	R/W
APP_MAC_INTR_MAP_REG	interrupt map	0x3FF00218	R/W
APP_MAC_NMI_MAP_REG	interrupt map	0x3FF0021C	R/W
APP_BB_INT_MAP_REG	interrupt map	0x3FF00220	R/W
APP_BT_MAC_INT_MAP_REG	interrupt map	0x3FF00224	R/W
APP_BT_BB_INT_MAP_REG	interrupt map	0x3FF00228	R/W
APP_BT_BB_NMI_MAP_REG	interrupt map	0x3FF0022C	R/W
APP_RWBT_IRQ_MAP_REG	interrupt map	0x3FF00230	R/W
APP_RWBLE_IRQ_MAP_REG	interrupt map	0x3FF00234	R/W
APP_RWBT_NMI_MAP_REG	interrupt map	0x3FF00238	R/W
APP_RWBLE_NMI_MAP_REG	interrupt map	0x3FF0023C	R/W
APP_SLC0_INTR_MAP_REG	interrupt map	0x3FF00240	R/W
APP_SLC1_INTR_MAP_REG	interrupt map	0x3FF00244	R/W
APP_UHCI0_INTR_MAP_REG	interrupt map	0x3FF00248	R/W
APP_UHCI1_INTR_MAP_REG	interrupt map	0x3FF0024C	R/W
APP_TG_T0_LEVEL_INT_MAP_REG	interrupt map	0x3FF00250	R/W
APP_TG_T1_LEVEL_INT_MAP_REG	interrupt map	0x3FF00254	R/W
APP_TG_WDT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00258	R/W
APP_TG_LACT_LEVEL_INT_MAP_REG	interrupt map	0x3FF0025C	R/W
APP_TG1_T0_LEVEL_INT_MAP_REG	interrupt map	0x3FF00260	R/W
APP_TG1_T1_LEVEL_INT_MAP_REG	interrupt map	0x3FF00264	R/W
APP_TG1_WDT_LEVEL_INT_MAP_REG	interrupt map	0x3FF00268	R/W
APP_TG1_LACT_LEVEL_INT_MAP_REG	interrupt map	0x3FF0026C	R/W
APP_GPIO_INTERRUPT_MAP_REG	interrupt map	0x3FF00270	R/W
APP_GPIO_INTERRUPT_NMI_MAP_REG	interrupt map	0x3FF00274	R/W

Name	Description	Address	Access
APP_CPU_INTR_FROM_CPU_0_MAP_REG	interrupt map	0x3FF00278	R/W
APP_CPU_INTR_FROM_CPU_1_MAP_REG	interrupt map	0x3FF0027C	R/W
APP_CPU_INTR_FROM_CPU_2_MAP_REG	interrupt map	0x3FF00280	R/W
APP_CPU_INTR_FROM_CPU_3_MAP_REG	interrupt map	0x3FF00284	R/W
APP_SPI_INTR_0_MAP_REG	interrupt map	0x3FF00288	R/W
APP_SPI_INTR_1_MAP_REG	interrupt map	0x3FF0028C	R/W
APP_SPI_INTR_2_MAP_REG	interrupt map	0x3FF00290	R/W
APP_SPI_INTR_3_MAP_REG	interrupt map	0x3FF00294	R/W
APP_I2S0_INT_MAP_REG	interrupt map	0x3FF00298	R/W
APP_I2S1_INT_MAP_REG	interrupt map	0x3FF0029C	R/W
APP_UART_INTR_MAP_REG	interrupt map	0x3FF002A0	R/W
APP_UART1_INTR_MAP_REG	interrupt map	0x3FF002A4	R/W
APP_UART2_INTR_MAP_REG	interrupt map	0x3FF002A8	R/W
APP_SDIO_HOST_INTERRUPT_MAP_REG	interrupt map	0x3FF002AC	R/W
APP_EMAC_INT_MAP_REG	interrupt map	0x3FF002B0	R/W
APP_PWM0_INTR_MAP_REG	interrupt map	0x3FF002B4	R/W
APP_PWM1_INTR_MAP_REG	interrupt map	0x3FF002B8	R/W
APP_PWM2_INTR_MAP_REG	interrupt map	0x3FF002BC	R/W
APP_PWM3_INTR_MAP_REG	interrupt map	0x3FF002C0	R/W
APP_LEDC_INT_MAP_REG	interrupt map	0x3FF002C4	R/W
APP_EFUSE_INT_MAP_REG	interrupt map	0x3FF002C8	R/W
APP_CAN_INT_MAP_REG	interrupt map	0x3FF002CC	R/W
APP_RTC_CORE_INTR_MAP_REG	interrupt map	0x3FF002D0	R/W
APP_RMT_INTR_MAP_REG	interrupt map	0x3FF002D4	R/W
APP_PCNT_INTR_MAP_REG	interrupt map	0x3FF002D8	R/W
APP_I2C_EXT0_INTR_MAP_REG	interrupt map	0x3FF002DC	R/W
APP_I2C_EXT1_INTR_MAP_REG	interrupt map	0x3FF002E0	R/W
APP_RSA_INTR_MAP_REG	interrupt map	0x3FF002E4	R/W
APP_SPI1_DMA_INT_MAP_REG	interrupt map	0x3FF002E8	R/W
APP_SPI2_DMA_INT_MAP_REG	interrupt map	0x3FF002EC	R/W
APP_SPI3_DMA_INT_MAP_REG	interrupt map	0x3FF002F0	R/W
APP_WDG_INT_MAP_REG	interrupt map	0x3FF002F4	R/W
APP_TIMER_INT1_MAP_REG	interrupt map	0x3FF002F8	R/W
APP_TIMER_INT2_MAP_REG	interrupt map	0x3FF002FC	R/W
APP_TG_T0_EDGE_INT_MAP_REG	interrupt map	0x3FF00300	R/W
APP_TG_T1_EDGE_INT_MAP_REG	interrupt map	0x3FF00304	R/W
APP_TG_WDT_EDGE_INT_MAP_REG	interrupt map	0x3FF00308	R/W
APP_TG_LACT_EDGE_INT_MAP_REG	interrupt map	0x3FF0030C	R/W
APP_TG1_T0_EDGE_INT_MAP_REG	interrupt map	0x3FF00310	R/W
APP_TG1_T1_EDGE_INT_MAP_REG	interrupt map	0x3FF00314	R/W
APP_TG1_WDT_EDGE_INT_MAP_REG	interrupt map	0x3FF00318	R/W
APP_TG1_LACT_EDGE_INT_MAP_REG	interrupt map	0x3FF0031C	R/W
APP_MMU_IA_INT_MAP_REG	interrupt map	0x3FF00320	R/W
APP_MPU_IA_INT_MAP_REG	interrupt map	0x3FF00324	R/W

Name	Description	Address	Access
APP_CACHE_IA_INT_MAP_REG	interrupt map	0x3FF00328	R/W
AHBLITE_MPU_TABLE_UART_REG	MPU for peripherals	0x3FF0032C	R/W
AHBLITE_MPU_TABLE_SPI1_REG	MPU for peripherals	0x3FF00330	R/W
AHBLITE_MPU_TABLE_SPI0_REG	MPU for peripherals	0x3FF00334	R/W
AHBLITE_MPU_TABLE_GPIO_REG	MPU for peripherals	0x3FF00338	R/W
AHBLITE_MPU_TABLE_RTC_REG	MPU for peripherals	0x3FF00348	R/W
AHBLITE_MPU_TABLE_IO_MUX_REG	MPU for peripherals	0x3FF0034C	R/W
AHBLITE_MPU_TABLE_HINF_REG	MPU for peripherals	0x3FF00354	R/W
AHBLITE_MPU_TABLE_UHCI1_REG	MPU for peripherals	0x3FF00358	R/W
AHBLITE_MPU_TABLE_I2S0_REG	MPU for peripherals	0x3FF00364	R/W
AHBLITE_MPU_TABLE_UART1_REG	MPU for peripherals	0x3FF00368	R/W
AHBLITE_MPU_TABLE_I2C_EXT0_REG	MPU for peripherals	0x3FF00374	R/W
AHBLITE_MPU_TABLE_UHCI0_REG	MPU for peripherals	0x3FF00378	R/W
AHBLITE_MPU_TABLE_SLCHOST_REG	MPU for peripherals	0x3FF0037C	R/W
AHBLITE_MPU_TABLE_RMT_REG	MPU for peripherals	0x3FF00380	R/W
AHBLITE_MPU_TABLE_PCNT_REG	MPU for peripherals	0x3FF00384	R/W
AHBLITE_MPU_TABLE_SLC_REG	MPU for peripherals	0x3FF00388	R/W
AHBLITE_MPU_TABLE_LEDC_REG	MPU for peripherals	0x3FF0038C	R/W
AHBLITE_MPU_TABLE_EFUSE_REG	MPU for peripherals	0x3FF00390	R/W
AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG	MPU for peripherals	0x3FF00394	R/W
AHBLITE_MPU_TABLE_PWM0_REG	MPU for peripherals	0x3FF0039C	R/W
AHBLITE_MPU_TABLE_TIMERGROUP_REG	MPU for peripherals	0x3FF003A0	R/W
AHBLITE_MPU_TABLE_TIMERGROUP1_REG	MPU for peripherals	0x3FF003A4	R/W
AHBLITE_MPU_TABLE_SPI2_REG	MPU for peripherals	0x3FF003A8	R/W
AHBLITE_MPU_TABLE_SPI3_REG	MPU for peripherals	0x3FF003AC	R/W
AHBLITE_MPU_TABLE_APB_CTRL_REG	MPU for peripherals	0x3FF003B0	R/W
AHBLITE_MPU_TABLE_I2C_EXT1_REG	MPU for peripherals	0x3FF003B4	R/W
AHBLITE_MPU_TABLE_SDIO_HOST_REG	MPU for peripherals	0x3FF003B8	R/W
AHBLITE_MPU_TABLE_EMAC_REG	MPU for peripherals	0x3FF003BC	R/W
AHBLITE_MPU_TABLE_PWM1_REG	MPU for peripherals	0x3FF003C4	R/W
AHBLITE_MPU_TABLE_I2S1_REG	MPU for peripherals	0x3FF003C8	R/W
AHBLITE_MPU_TABLE_UART2_REG	MPU for peripherals	0x3FF003CC	R/W
AHBLITE_MPU_TABLE_PWM2_REG	MPU for peripherals	0x3FF003D0	R/W
AHBLITE_MPU_TABLE_PWM3_REG	MPU for peripherals	0x3FF003D4	R/W
AHBLITE_MPU_TABLE_PWR_REG	MPU for peripherals	0x3FF003E4	R/W
IMMU_TABLE0_REG	MMU register 1 for internal SRAM 0	0x3FF00504	R/W
IMMU_TABLE1_REG	MMU register 1 for internal SRAM 0	0x3FF00508	R/W
IMMU_TABLE2_REG	MMU register 1 for Internal SRAM 0	0x3FF0050C	R/W
IMMU_TABLE3_REG	MMU register 1 for internal SRAM 0	0x3FF00510	R/W
IMMU_TABLE4_REG	MMU register 1 for internal SRAM 0	0x3FF00514	R/W
IMMU_TABLE5_REG	MMU register 1 for internal SRAM 0	0x3FF00518	R/W
IMMU_TABLE6_REG	MMU register 1 for internal SRAM 0	0x3FF0051C	R/W
IMMU_TABLE7_REG	MMU register 1 for internal SRAM 0	0x3FF00520	R/W
IMMU_TABLE8_REG	MMU register 1 for internal SRAM 0	0x3FF00524	R/W

Name	Description	Address	Access
IMMU_TABLE9_REG	MMU register 1 for internal SRAM 0	0x3FF00528	R/W
IMMU_TABLE10_REG	MMU register 1 for internal SRAM 0	0x3FF0052C	R/W
IMMU_TABLE11_REG	MMU register 1 for internal SRAM 0	0x3FF00530	R/W
IMMU_TABLE12_REG	MMU register 1 for Internal SRAM 0	0x3FF00534	R/W
IMMU_TABLE13_REG	MMU register 1 for internal SRAM 0	0x3FF00538	R/W
IMMU_TABLE14_REG	MMU register 1 for internal SRAM 0	0x3FF0053C	R/W
IMMU_TABLE15_REG	MMU register 1 for internal SRAM 0	0x3FF00540	R/W
DMMU_TABLE0_REG	MMU register 1 for Internal SRAM 2	0x3FF00544	R/W
DMMU_TABLE1_REG	MMU register 1 for internal SRAM 2	0x3FF00548	R/W
DMMU_TABLE2_REG	MMU register 1 for internal SRAM 2	0x3FF0054C	R/W
DMMU_TABLE3_REG	MMU register 1 for internal SRAM 2	0x3FF00550	R/W
DMMU_TABLE4_REG	MMU register 1 for internal SRAM 2	0x3FF00554	R/W
DMMU_TABLE5_REG	MMU register 1 for internal SRAM 2	0x3FF00558	R/W
DMMU_TABLE6_REG	MMU register 1 for internal SRAM 2	0x3FF0055C	R/W
DMMU_TABLE7_REG	MMU register 1 for internal SRAM 2	0x3FF00560	R/W
DMMU_TABLE8_REG	MMU register 1 for internal SRAM 2	0x3FF00564	R/W
DMMU_TABLE9_REG	MMU register 1 for internal SRAM 2	0x3FF00568	R/W
DMMU_TABLE10_REG	MMU register 1 for internal SRAM 2	0x3FF0056C	R/W
DMMU_TABLE11_REG	MMU register 1 for internal SRAM 2	0x3FF00570	R/W
DMMU_TABLE12_REG	MMU register 1 for internal SRAM 2	0x3FF00574	R/W
DMMU_TABLE13_REG	MMU register 1 for internal SRAM 2	0x3FF00578	R/W
DMMU_TABLE14_REG	MMU register 1 for internal SRAM 2	0x3FF0057C	R/W
DMMU_TABLE15_REG	MMU register 1 for internal SRAM 2	0x3FF00580	R/W
SECURE_BOOT_CTRL_REG	mode for secure_boot	0x3FF005A4	R/W
SPI_DMA_CHAN_SEL_REG	selects DMA channel for SPI1, SPI2, and SPI3	0x3FF005A8	R/W

5.5 Registers

Register 5.1: PRO_BOOT_REMAP_CTRL_REG (0x000)

(reserved)	PRO_BOOT_REMAP	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PRO_BOOT_REMAP Remap mode for PRO_CPU. (R/W)

Register 5.2: APP_BOOT_REMAP_CTRL_REG (0x004)

(reserved)	APP_BOOT_REMAP	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

APP_BOOT_REMAP Remap mode for APP_CPU. (R/W)

Register 5.3: PERI_CLK_EN_REG (0x01C)

31 0	
0x000000000	Reset

PERI_CLK_EN_REG Clock gate for peripherals. (R/W)

Register 5.4: PERI_RST_EN_REG (0x020)

31 0	
0x000000000	Reset

PERI_RST_EN_REG Reset for peripherals. (R/W)

Register 5.5: APPCPU_CTRL_REG_A_REG (0x02C)

(reserved)	APPCPU_RESETTING	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

APPCPU_RESETTING Reset for APP_CPU. (R/W)

Register 5.6: APPCPU_CTRL_REG_B_REG (0x030)

(reserved)	APPCPU_CLKGATE_EN	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

APPCPU_CLKGATE_EN Clock gate for APP_CPU. (R/W)

Register 5.7: APPCPU_CTRL_REG_C_REG (0x034)

(reserved)	APPCPU_RUNSTALL	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

APPCPU_RUNSTALL Stall for APP_CPU. (R/W)

Register 5.8: APPCPU_CTRL_REG_D_REG (0x038)

31 0	
0x000000000	Reset

APPCPU_CTRL_REG_D_REG Boot address for APP_CPU. (R/W)

Register 5.9: CPU_PER_CONF_REG (0x03C)

(reserved)	CPU_CPUPERIOD_SEL	
31 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	Reset

CPU_CPUPERIOD_SEL Select CPU clock. (R/W)

Register 5.10: PRO_CACHE_CTRL_REG (0x040)

(reserved)	PRO_DRAM_HL	(reserved)	PRO_DRAM_SPLIT	PRO_SINGLE_IRAM_ENA	(reserved)	PRO_CACHE_FLUSH_DONE	PRO_CACHE_FLUSH_ENA	PRO_CACHE_ENABLE	(reserved)	
31 17	16	15 12	11	10	9 6	5	4	3	5 3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0	0	0	0 0 0 0	0	1	0	0 0 0	Reset

PRO_DRAM_HL Determines the virtual address mode of the external SRAM. (R/W)

PRO_DRAM_SPLIT Determines the virtual address mode of the external SRAM. (R/W)

PRO_SINGLE_IRAM_ENA Determines a special mode for PRO_CPU access to the external flash. (R/W)

PRO_CACHE_FLUSH_DONE PRO_CPU cache-flush done. (RO)

PRO_CACHE_FLUSH_ENA Flushes the PRO_CPU cache. (R/W)

PRO_CACHE_ENABLE Enables the PRO_CPU cache. (R/W)

Register 5.11: APP_CACHE_CTRL_REG (0x058)

(reserved)	APP_DRAM_HL	(reserved)	APP_DRAM_SPLIT	APP_SINGLE_IRAM_ENA	(reserved)	APP_CACHE_FLUSH_DONE	APP_CACHE_FLUSH_ENA	APP_CACHE_ENABLE	(reserved)	
31 15	14	13 12	11	10	9 6	5	4	3	5 3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0	0	0	0 0 0 0	0	1	0	0 0 0	Reset

APP_DRAM_HL Determines the virtual address mode of the External SRAM. (R/W)

APP_DRAM_SPLIT Determines the virtual address mode of the External SRAM. (R/W)

APP_SINGLE_IRAM_ENA Determines a special mode for APP_CPU access to the external flash. (R/W)

APP_CACHE_FLUSH_DONE APP_CPU cache-flush done. (RO)

APP_CACHE_FLUSH_ENA Flushes the APP_CPU cache. (R/W)

APP_CACHE_ENABLE Enables the APP_CPU cache. (R/W)

Register 5.12: CACHE_MUX_MODE_REG (0x07C)

(reserved)	CACHE_MUX_MODE	
31 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	Reset

CACHE_MUX_MODE The mode of the two caches sharing the memory. (R/W)

Register 5.13: IMMU_PAGE_MODE_REG (0x080)

(reserved)	IMMU_PAGE_MODE	(reserved)	
31 3	2 1	1	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0	Reset

IMMU_PAGE_MODE Page size in the MMU for the internal SRAM 0. (R/W)

Register 5.14: DMMU_PAGE_MODE_REG (0x084)

(reserved)	DMMU_PAGE_MODE	(reserved)	
31 3	2 1	1	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0	Reset

DMMU_PAGE_MODE Page size in the MMU for the internal SRAM 2. (R/W)

Register 5.15: SRAM_PD_CTRL_REG_0_REG (0x098)

31 0	
0x000000000	Reset

SRAM_PD_CTRL_REG_0_REG Powers down the internal SRAM. (R/W)

Register 5.16: SRAM_PD_CTRL_REG_1_REG (0x09C)

(reserved)	SRAM_PD_1	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SRAM_PD_1 Powers down the internal SRAM. (R/W)

Register 5.17: AHB_MPU_TABLE_0_REG (0x0B4)

31 0	
0x0FFFFFFFF	Reset

AHB_MPU_TABLE_0_REG MPU for DMA. (R/W)

Register 5.18: AHB_MPU_TABLE_1_REG (0x0B8)

(reserved)	AHB_ACCESS_GRANT_1	
31 9	8 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x1FF	Reset

AHB_ACCESS_GRANT_1 MPU for DMA. (R/W)

Register 5.19: PERIP_CLK_EN_REG (0x0C0)

31 0	
0x0F9C1E06F	Reset

PERIP_CLK_EN_REG Clock gate for peripherals. (R/W)

Register 5.20: PERIP_RST_EN_REG (0x0C4)

31 0	
0x000000000	Reset

PERIP_RST_EN_REG Reset for peripherals. (R/W)

Register 5.21: SLAVE_SPI_CONFIG_REG (0x0C8)

(reserved)	SLAVE_SPI_DECRYPT_ENABLE	(reserved)	SLAVE_SPI_ENCRYPT_ENABLE	(reserved)	
31 13	12	11 9	8	15 8	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0	0	0 0 0 0 0 0 0 0	Reset

SLAVE_SPI_DECRYPT_ENABLE Enables decryption in the external flash. (R/W)

SLAVE_SPI_ENCRYPT_ENABLE Enables encryption in the external flash. (R/W)

Register 5.22: WIFI_CLK_EN_REG (0x0CC)

31	0	
0x0FFFCE030		Reset

WIFI_CLK_EN_REG Clock gate for Wi-Fi. (R/W)

Register 5.23: WIFI_RST_EN_REG (0x0D0)

31	0	
0x000000000		Reset

WIFI_RST_EN_REG Reset for Wi-Fi. (R/W)

Register 5.24: CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3) (0xDC+4*n)

(reserved)		CPU_INTR_FROM_CPU_*n*	
31	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0	Reset

CPU_INTR_FROM_CPU_*n* Interrupt in both CPUs. (R/W)

Register 5.25: PRO_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xEC+4**n*)

31	0	
0x000000000		Reset

PRO_INTR_STATUS_REG_*n*_REG PRO_CPU interrupt status. (RO)

Register 5.26: APP_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xF8+4**n*)

31	0	
0x000000000		Reset

APP_INTR_STATUS_REG_*n*_REG APP_CPU interrupt status. (RO)

Register 5.27: PRO_MAC_INTR_MAP_REG (0x104)

Register 5.28: PRO_MAC_NMI_MAP_REG (0x108)

Register 5.29: PRO_BB_INT_MAP_REG (0x10C)

Register 5.30: PRO_BT_MAC_INT_MAP_REG (0x110)

Register 5.31: PRO_BT_BB_INT_MAP_REG (0x114)

Register 5.32: PRO_BT_BB_NMI_MAP_REG (0x118)

Register 5.33: PRO_RWBT_IRQ_MAP_REG (0x11C)

Register 5.34: PRO_RWBLE_IRQ_MAP_REG (0x120)

Register 5.35: PRO_RWBT_NMI_MAP_REG (0x124)

Register 5.36: PRO_RWBLE_NMI_MAP_REG (0x128)

Register 5.37: PRO_SLC0_INTR_MAP_REG (0x12C)

Register 5.38: PRO_SLC1_INTR_MAP_REG (0x130)

Register 5.39: PRO_UHCI0_INTR_MAP_REG (0x134)

Register 5.40: PRO_UHCI1_INTR_MAP_REG (0x138)

Register 5.41: PRO_TG_T0_LEVEL_INT_MAP_REG (0x13C)

Register 5.42: PRO_TG_T1_LEVEL_INT_MAP_REG (0x140)

Register 5.43: PRO_TG_WDT_LEVEL_INT_MAP_REG (0x144)

Register 5.44: PRO_TG_LACT_LEVEL_INT_MAP_REG (0x148)

Register 5.45: PRO_TG1_T0_LEVEL_INT_MAP_REG (0x14C)

Register 5.46: PRO_TG1_T1_LEVEL_INT_MAP_REG (0x150)

Register 5.47: PRO_TG1_WDT_LEVEL_INT_MAP_REG (0x154)

Register 5.48: PRO_TG1_LACT_LEVEL_INT_MAP_REG (0x158)

Register 5.49: PRO_GPIO_INTERRUPT_MAP_REG (0x15C)

Register 5.50: PRO_GPIO_INTERRUPT_NMI_MAP_REG (0x160)

Register 5.51: PRO_CPU_INTR_FROM_CPU_0_MAP_REG (0x164)

Register 5.52: PRO_CPU_INTR_FROM_CPU_1_MAP_REG (0x168)

Register 5.53: PRO_CPU_INTR_FROM_CPU_2_MAP_REG (0x16C)

Register 5.54: PRO_CPU_INTR_FROM_CPU_3_MAP_REG (0x170)

Register 5.55: PRO_SPI_INTR_0_MAP_REG (0x174)

Register 5.56: PRO_SPI_INTR_1_MAP_REG (0x178)

Register 5.57: PRO_SPI_INTR_2_MAP_REG (0x17C)

Register 5.58: PRO_SPI_INTR_3_MAP_REG (0x180)

Register 5.59: PRO_I2S0_INT_MAP_REG (0x184)

Register 5.60: PRO_I2S1_INT_MAP_REG (0x188)

Register 5.61: PRO_UART_INTR_MAP_REG (0x18C)

Register 5.62: PRO_UART1_INTR_MAP_REG (0x190)

Register 5.63: PRO_UART2_INTR_MAP_REG (0x194)

Register 5.64: PRO_SDIO_HOST_INTERRUPT_MAP_REG (0x198)

Register 5.65: PRO_EMAC_INT_MAP_REG (0x19C)

Register 5.66: PRO_PWM0_INTR_MAP_REG (0x1A0)

Register 5.67: PRO_PWM1_INTR_MAP_REG (0x1A4)

Register 5.68: PRO_PWM2_INTR_MAP_REG (0x1A8)

Register 5.69: PRO_PWM3_INTR_MAP_REG (0x1AC)

Register 5.70: PRO_LEDC_INT_MAP_REG (0x1B0)

Register 5.71: PRO_EFUSE_INT_MAP_REG (0x1B4)

Register 5.72: PRO_CAN_INT_MAP_REG (0x1B8)

Register 5.73: PRO_RTC_CORE_INTR_MAP_REG (0x1BC)

Register 5.74: PRO_RMT_INTR_MAP_REG (0x1C0)

Register 5.75: PRO_PCNT_INTR_MAP_REG (0x1C4)

Register 5.76: PRO_I2C_EXT0_INTR_MAP_REG (0x1C8)

Register 5.77: PRO_I2C_EXT1_INTR_MAP_REG (0x1CC)

Register 5.78: PRO_RSA_INTR_MAP_REG (0x1D0)

Register 5.79: PRO_SPI1_DMA_INT_MAP_REG (0x1D4)

Register 5.80: PRO_SPI2_DMA_INT_MAP_REG (0x1D8)

Register 5.81: PRO_SPI3_DMA_INT_MAP_REG (0x1DC)

Register 5.82: PRO_WDG_INT_MAP_REG (0x1E0)

Register 5.83: PRO_TIMER_INT1_MAP_REG (0x1E4)

Register 5.84: PRO_TIMER_INT2_MAP_REG (0x1E8)

Register 5.85: PRO_TG_T0_EDGE_INT_MAP_REG (0x1EC)

Register 5.86: PRO_TG_T1_EDGE_INT_MAP_REG (0x1F0)

Register 5.87: PRO_TG_WDT_EDGE_INT_MAP_REG (0x1F4)

Register 5.88: PRO_TG_LACT_EDGE_INT_MAP_REG (0x1F8)

Register 5.89: PRO_TG1_T0_EDGE_INT_MAP_REG (0x1FC)

Register 5.90: PRO_TG1_T1_EDGE_INT_MAP_REG (0x200)

Register 5.91: PRO_TG1_WDT_EDGE_INT_MAP_REG (0x204)

Register 5.92: PRO_TG1_LACT_EDGE_INT_MAP_REG (0x208)

Register 5.93: PRO_MMU_IA_INT_MAP_REG (0x20C)

Register 5.94: PRO_MPU_IA_INT_MAP_REG (0x210)

Register 5.95: PRO_CACHE_IA_INT_MAP_REG (0x214)

(reserved)	PRO_*_MAP	
31 5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

PRO_*_MAP Interrupt map. (R/W)

Register 5.96: APP_MAC_INTR_MAP_REG (0x218)

Register 5.97: APP_MAC_NMI_MAP_REG (0x21C)

Register 5.98: APP_BB_INT_MAP_REG (0x220)

Register 5.99: APP_BT_MAC_INT_MAP_REG (0x224)

Register 5.100: APP_BT_BB_INT_MAP_REG (0x228)

Register 5.101: APP_BT_BB_NMI_MAP_REG (0x22C)

Register 5.102: APP_RWBT_IRQ_MAP_REG (0x230)

Register 5.103: APP_RWBLE_IRQ_MAP_REG (0x234)

Register 5.104: APP_RWBT_NMI_MAP_REG (0x238)

Register 5.105: APP_RWBLE_NMI_MAP_REG (0x23C)

Register 5.106: APP_SLC0_INTR_MAP_REG (0x240)

Register 5.107: APP_SLC1_INTR_MAP_REG (0x244)

Register 5.108: APP_UHCI0_INTR_MAP_REG (0x248)

Register 5.109: APP_UHCI1_INTR_MAP_REG (0x24C)

Register 5.110: APP_TG_T0_LEVEL_INT_MAP_REG (0x250)

Register 5.111: APP_TG_T1_LEVEL_INT_MAP_REG (0x254)

Register 5.112: APP_TG_WDT_LEVEL_INT_MAP_REG (0x258)

Register 5.113: APP_TG_LACT_LEVEL_INT_MAP_REG (0x25C)

Register 5.114: APP_TG1_T0_LEVEL_INT_MAP_REG (0x260)

Register 5.115: APP_TG1_T1_LEVEL_INT_MAP_REG (0x264)

Register 5.116: APP_TG1_WDT_LEVEL_INT_MAP_REG (0x268)

Register 5.117: APP_TG1_LACT_LEVEL_INT_MAP_REG (0x26C)

Register 5.118: APP_GPIO_INTERRUPT_MAP_REG (0x270)

Register 5.119: APP_GPIO_INTERRUPT_NMI_MAP_REG (0x274)

Register 5.120: APP_CPU_INTR_FROM_CPU_0_MAP_REG (0x278)

Register 5.121: APP_CPU_INTR_FROM_CPU_1_MAP_REG (0x27C)

Register 5.122: APP_CPU_INTR_FROM_CPU_2_MAP_REG (0x280)

Register 5.123: APP_CPU_INTR_FROM_CPU_3_MAP_REG (0x284)

Register 5.124: APP_SPI_INTR_0_MAP_REG (0x288)

Register 5.125: APP_SPI_INTR_1_MAP_REG (0x28C)

Register 5.126: APP_SPI_INTR_2_MAP_REG (0x290)

Register 5.127: APP_SPI_INTR_3_MAP_REG (0x294)

Register 5.128: APP_I2S0_INT_MAP_REG (0x298)

Register 5.129: APP_I2S1_INT_MAP_REG (0x29C)

Register 5.130: APP_UART_INTR_MAP_REG (0x2A0)

Register 5.131: APP_UART1_INTR_MAP_REG (0x2A4)

Register 5.132: APP_UART2_INTR_MAP_REG (0x2A8)

Register 5.133: APP_SDIO_HOST_INTERRUPT_MAP_REG (0x2AC)

Register 5.134: APP_EMAC_INT_MAP_REG (0x2B0)

Register 5.135: APP_PWM0_INTR_MAP_REG (0x2B4)

Register 5.136: APP_PWM1_INTR_MAP_REG (0x2B8)

Register 5.137: APP_PWM2_INTR_MAP_REG (0x2BC)

Register 5.138: APP_PWM3_INTR_MAP_REG (0x2C0)

Register 5.139: APP_LEDC_INT_MAP_REG (0x2C4)

Register 5.140: APP_EFUSE_INT_MAP_REG (0x2C8)

Register 5.141: APP_CAN_INT_MAP_REG (0x2CC)

Register 5.142: APP_RTC_CORE_INTR_MAP_REG (0x2D0)

Register 5.143: APP_RMT_INTR_MAP_REG (0x2D4)

Register 5.144: APP_PCNT_INTR_MAP_REG (0x2D8)

Register 5.145: APP_I2C_EXT0_INTR_MAP_REG (0x2DC)

Register 5.146: APP_I2C_EXT1_INTR_MAP_REG (0x2E0)

Register 5.147: APP_RSA_INTR_MAP_REG (0x2E4)

Register 5.148: APP_SPI1_DMA_INT_MAP_REG (0x2E8)

Register 5.149: APP_SPI2_DMA_INT_MAP_REG (0x2EC)

Register 5.150: APP_SPI3_DMA_INT_MAP_REG (0x2F0)

Register 5.151: APP_WDG_INT_MAP_REG (0x2F4)

Register 5.152: APP_TIMER_INT1_MAP_REG (0x2F8)

Register 5.153: APP_TIMER_INT2_MAP_REG (0x2FC)

Register 5.154: APP_TG_T0_EDGE_INT_MAP_REG (0x300)

Register 5.155: APP_TG_T1_EDGE_INT_MAP_REG (0x304)

Register 5.156: APP_TG_WDT_EDGE_INT_MAP_REG (0x308)

Register 5.157: APP_TG_LACT_EDGE_INT_MAP_REG (0x30C)

Register 5.158: APP_TG1_T0_EDGE_INT_MAP_REG (0x310)

Register 5.159: APP_TG1_T1_EDGE_INT_MAP_REG (0x314)

Register 5.160: APP_TG1_WDT_EDGE_INT_MAP_REG (0x318)

Register 5.161: APP_TG1_LACT_EDGE_INT_MAP_REG (0x31C)

Register 5.162: APP_MMU_IA_INT_MAP_REG (0x320)

Register 5.163: APP_MPU_IA_INT_MAP_REG (0x324)

Register 5.164: APP_CACHE_IA_INT_MAP_REG (0x328)

(reserved)	APP_*_MAP	
31 5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	16	Reset

APP_*_MAP Interrupt map. (R/W)

Register 5.165: AHBLITE_MPU_TABLE_UART_REG (0x32C)

Register 5.166: AHBLITE_MPU_TABLE_SPI1_REG (0x330)

Register 5.167: AHBLITE_MPU_TABLE_SPI0_REG (0x334)

Register 5.168: AHBLITE_MPU_TABLE_GPIO_REG (0x338)

Register 5.169: AHBLITE_MPU_TABLE_RTC_REG (0x348)

Register 5.170: AHBLITE_MPU_TABLE_IO_MUX_REG (0x34C)

Register 5.171: AHBLITE_MPU_TABLE_HINF_REG (0x354)

Register 5.172: AHBLITE_MPU_TABLE_UHCI1_REG (0x358)

Register 5.173: AHBLITE_MPU_TABLE_I2S0_REG (0x364)

Register 5.174: AHBLITE_MPU_TABLE_UART1_REG (0x368)

Register 5.175: AHBLITE_MPU_TABLE_I2C_EXT0_REG (0x374)

Register 5.176: AHBLITE_MPU_TABLE_UHCI0_REG (0x378)

Register 5.177: AHBLITE_MPU_TABLE_SLCHOST_REG (0x37C)

Register 5.178: AHBLITE_MPU_TABLE_RMT_REG (0x380)

Register 5.179: AHBLITE_MPU_TABLE_PCNT_REG (0x384)

Register 5.180: AHBLITE_MPU_TABLE_SLC_REG (0x388)

Register 5.181: AHBLITE_MPU_TABLE_LEDC_REG (0x38C)

Register 5.182: AHBLITE_MPU_TABLE_EFUSE_REG (0x390)

Register 5.183: AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG (0x394)

Register 5.184: AHBLITE_MPU_TABLE_PWM0_REG (0x39C)

Register 5.185: AHBLITE_MPU_TABLE_TIMERGROUP_REG (0x3A0)

Register 5.186: AHBLITE_MPU_TABLE_TIMERGROUP1_REG (0x3A4)

Register 5.187: AHBLITE_MPU_TABLE_SPI2_REG (0x3A8)

Register 5.188: AHBLITE_MPU_TABLE_SPI3_REG (0x3AC)

Register 5.189: AHBLITE_MPU_TABLE_APB_CTRL_REG (0x3B0)

Register 5.190: AHBLITE_MPU_TABLE_I2C_EXT1_REG (0x3B4)

Register 5.191: AHBLITE_MPU_TABLE_SDIO_HOST_REG (0x3B8)

Register 5.192: AHBLITE_MPU_TABLE_EMAC_REG (0x3BC)

Register 5.193: AHBLITE_MPU_TABLE_PWM1_REG (0x3C4)

Register 5.194: AHBLITE_MPU_TABLE_I2S1_REG (0x3C8)

Register 5.195: AHBLITE_MPU_TABLE_UART2_REG (0x3CC)

Register 5.196: AHBLITE_MPU_TABLE_PWM2_REG (0x3D0)

Register 5.197: AHBLITE_MPU_TABLE_PWM3_REG (0x3D4)

Register 5.198: AHBLITE_MPU_TABLE_PWR_REG (0x3E4)

(reserved)	AHBLITE_*_ACCESS_GRANT_CONFIG	
31 … 6	5 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0	Reset

AHBLITE_*_ACCESS_GRANT_CONFIG MPU for peripherals. (R/W)

Register 5.199: IMMU_TABLE*n*_REG (*n*: 0-15) (0x504+4**n*)

(reserved)	IMMU_TABLE*n*	
31 … 7	6 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	15	Reset

IMMU_TABLE*n* MMU for internal SRAM. (R/W)

Register 5.200: DMMU_TABLE*n*_REG (*n*: 0-15) (0x544+4**n*)

(reserved)	DMMU_TABLE*n*	
31 … 7	6 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	15	Reset

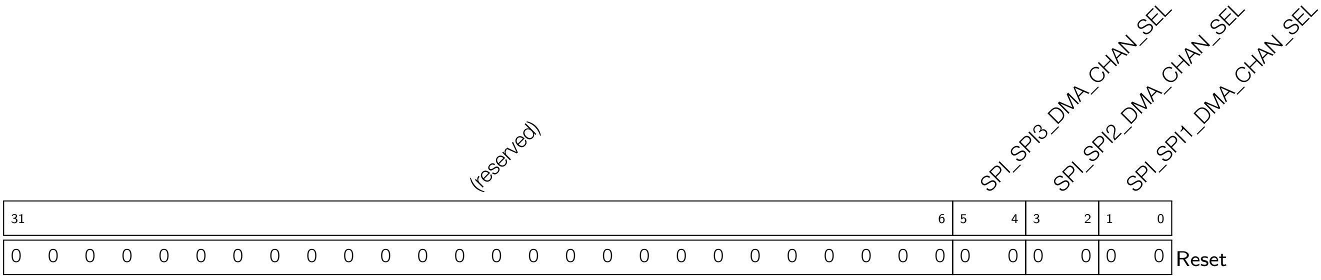
DMMU_TABLE*n* MMU for internal SRAM. (R/W)

Register 5.201: SECURE_BOOT_CTRL_REG (0x5A4)

(reserved)	SECURE_SW_BOOTLOADER_SEL	
31 … 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SECURE_SW_BOOTLOADER_SEL Mode for secure_boot. (R/W)

Register 5.202: SPI_DMA_CHAN_SEL_REG (0x5A8)



SPI_SPI3_DMA_CHAN_SEL Selects DMA channel for SPI3. (R/W)

SPI_SPI2_DMA_CHAN_SEL Selects DMA channel for SPI2. (R/W)

SPI_SPI1_DMA_CHAN_SEL Selects DMA channel for SPI1. (R/W)

6. DMA Controller

6.1 Overview

Direct Memory Access (DMA) is used for high-speed data transfer between peripherals and memory, as well as from memory to memory. Data can be quickly moved with DMA without any CPU intervention, thus allowing for more efficient use of the cores when processing data.

In the ESP32, 13 peripherals are capable of using DMA for data transfer, namely, UART0, UART1, UART2, SPI1, SPI2, SPI3, I2S0, I2S1, SDIO slave, SD/MMC host, EMAC, BT, and Wi-Fi.

6.2 Features

The DMA controllers in the ESP32 feature:

- AHB bus architecture
- Support for full-duplex and half-duplex data transfers
- Programmable data transfer length in bytes
- Support for 4-beat burst transfer
- 328 KB DMA address space
- All high-speed communication modules powered by DMA

6.3 Functional Description

All modules that require high-speed data transfer in bulk contain a DMA controller. DMA addressing uses the same data bus as the CPU to read/write to the internal RAM.

Each DMA controller features different functions. However, the architecture of the DMA engine (DMA_ENGINE) is the same in all DMA controllers.

6.3.1 DMA Engine Architecture

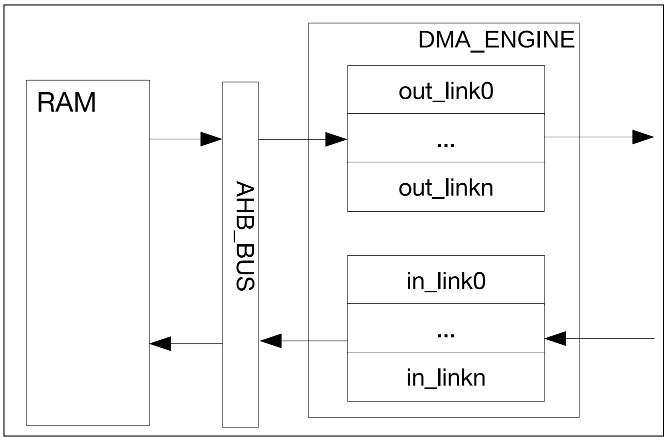


Figure 12: DMA Engine Architecture

The DMA Engine accesses SRAM over the AHB BUS. In Figure 12, the RAM represents the internal SRAM banks available on ESP32. Further details on the SRAM addressing range can be found in Chapter System and Memory. Software can use a DMA Engine by assigning a linked list to define the DMA operational parameters.

The DMA Engine transmits the data from the RAM to a peripheral, according to the contents of the out_link descriptor. Also, the DMA Engine stores the data received from a peripheral into a specified RAM location, according to the contents of the in_link descriptor.

6.3.2 Linked List

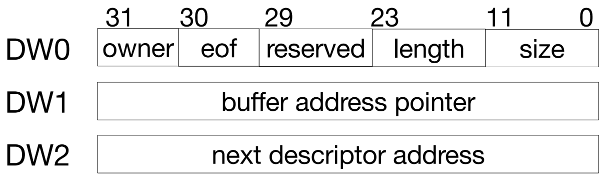


Figure 13: Linked List Structure

The DMA descriptor's linked lists (out_link and in_link) have the same structure. As shown in Figure 13, a linked-list descriptor consists of three words. The meaning of each field is as follows:

- owner (DW0) [31]: The allowed operator of the buffer corresponding to the current linked list.
  1'b0: the allowed operator is the CPU;
  1'b1: the allowed operator is the DMA controller.
- eof (DW0) [30]: End-Of-File character.
  1'b0: the linked-list item does not mark the end of the linked list;
  1'b1: the linked-list item is at the end of the linked list.
- reserved (DW0) [29:24]: Reserved bits.
  Software should not write 1's in this space.
- length (DW0) [23:12]: The number of valid bytes in the buffer corresponding to the current linked list. The field value indicates the number of bytes to be transferred to/from the buffer denoted by word DW1.
- size (DW0) [11:0]: The size of the buffer corresponding to the current linked list.
  NOTE: The size must be word-aligned.
- buffer address pointer (DW1): Buffer address pointer. This is the address of the data buffer.
  NOTE: The buffer address must be word-aligned.
- next descriptor address (DW2): The address pointer of the next linked-list item. The value is 0, if the current linked-list item is the last on the list (eof=1).

When receiving data, if the data transfer length is smaller than the specified buffer size, DMA will not use the remaining space. This enables the DMA engine to be used for transferring an arbitrary number of data bytes.

6.4 UART DMA (UDMA)

The ESP32 has three UART interfaces that share two UDMA (UART DMA) controllers. The UHCI*x*_UART_CE (*x* is 0 or 1) is used for selecting the UDMA.

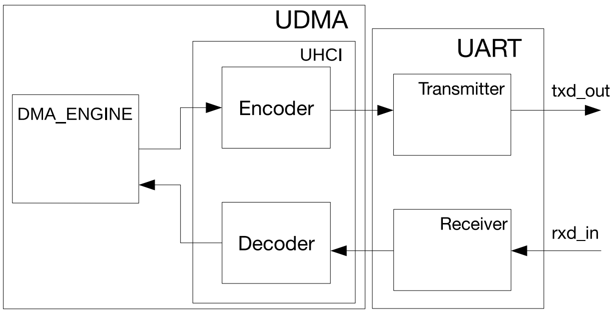


Figure 14: Data Transfer in UDMA Mode

Figure 14 shows the data transfer in UDMA mode. Before the DMA Engine receives data, software must initialize the receive-linked-list. UHCIx_INLINK_ADDR is used to point to the first in_link descriptor. The register must be programmed with the lower 20 bits of the address of the initial linked-list item. After UHCIx_INLINK_START is set, the Universal Host Controller Interface (UHCI) will transmit the data received by UART to the Decoder. After being parsed, the data will be stored in the RAM as specified by the receive-linked-list descriptor.

Before DMA transmits data, software must initialize the transmit-linked-list and the data to be transferred. UHCI_OUTLINK_ADDR is used to point to the first out_link descriptor. The register must be programmed with the lower 20 bits of the address of the initial transmit-linked-list item. After UHCIx_OUTLINK_START is set, the DMA Engine will read data from the RAM location specified by the linked-list descriptor and then transfer the data through the Encoder. The DMA Engine will then shift the data out serially through the UART transmitter.

The UART DMA follows a format of (separator + data + separator). The Encoder is used for adding separators before and after data, as well as using special-character sequences to replace data that are the same as separators. The Decoder is used for removing separators before and after data, as well as replacing the special-character sequences with separators. There can be multiple consecutive separators marking the beginning or end of data. These separators can be configured through UHCIx_SEPER_CH, with the default values being 0xC0. Data that are the same as separators can be replaced with UHCIx_ESC_SEQ0_CHAR0 (0xDB by default) and UHCIx_ESC_SEQ0_CHAR1 (0xDD by default). After the transmission process is complete, a UHCIx_OUT_TOTAL_EOF_INT interrupt will be generated. After the reception procedure is complete, a UHCIx_IN_SUC_EOF_INT interrupt will be generated.

6.5 SPI DMA Interface

ESP32 SPI modules can use DMA as well as the CPU for data exchange with peripherals. As can be seen from Figure 15, two DMA channels are shared by SPI1, SPI2 and SPI3 controllers. Each DMA channel can be used by any one SPI controller at any given time.

The ESP32 SPI DMA Engine also uses a linked list to receive/transmit data. Burst transmission is supported. The minimum data length for a single transfer is one byte. Consecutive data transfer is also supported.

SPI1_DMA_CHAN_SEL[1:0], SPI2_DMA_CHAN_SEL[1:0] and SPI3_DMA_CHAN_SEL[1:0] in DPORT_SPI_DMA_CHAN_SEL_REG must be configured to enable the SPI DMA interface for a specific SPI controller. Each SPI controller corresponds to one domain which has two bits with values 0, 1 and 2. Value 3 is reserved and must not be configured for operation.

Considering SPI1 as an example,
if SPI SPI1_DMA_CHAN_SEL[1:0] = 0, then SPI1 does not use any DMA channel;
if SPI1_DMA_CHAN_SEL[1:0] = 1, then SPI1 enables DMA channel1;

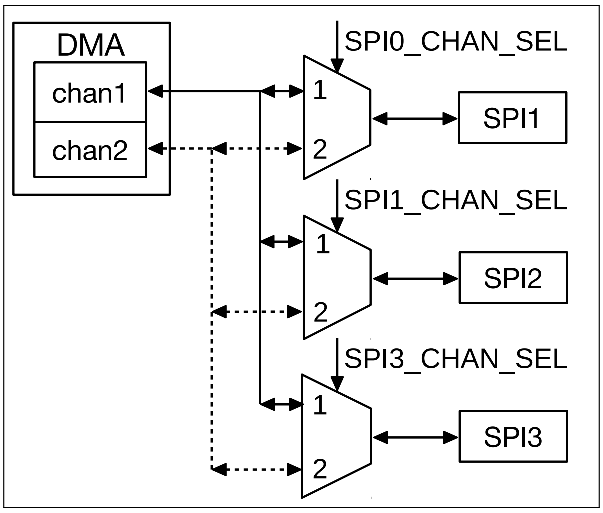


Figure 15: SPI DMA

if SPI1_DMA_CHAN_SEL[1:0] = 2, then SPI1 enables DMA channel2.

The SPI_OUTLINK_START bit in SPI_DMA_OUT_LINK_REG and the SPI_INLINK_START bit in SPI_DMA_IN_LINK_REG are used for enabling the DMA Engine. The two bits are self-cleared by hardware. When SPI_OUTLINK_START is set to 1, the DMA Engine starts processing the outbound linked list descriptor and prepares to transmit data. When SPI_INLINK_START is set to 1, then the DMA Engine starts processing the inbound linked-list descriptor and gets prepared to receive data.

Software should configure the SPI DMA as follows:

1. Reset the DMA state machine and FIFO parameters;
2. Configure the DMA-related registers for operation;
3. Configure the SPI-controller-related registers accordingly;
4. Set SPI_USR to enable DMA operation.

6.6 I2S DMA Interface

The ESP32 integrates two I2S modules, I2S0 and I2S1, each of which is powered by a DMA channel. The REG_I2S_DSCR_EN bit in I2S_FIFO_CONF_REG is used for enabling the DMA operation. ESP32 I2S DMA uses the standard linked-list descriptor to configure DMA operations for data transfer. Burst transfer is supported. However, unlike the SPI DMA channels, the data size for a single transfer is one word, or four bytes. REG_I2S_RX_EOF_NUM[31:0] bit in I2S_RXEOF_NUM_REG is used for configuring the data size of a single transfer operation, in multiples of one word.

I2S_OUTLINK_START bit in I2S_OUT_LINK_REG and I2S_INLINK_START bit in I2S_IN_LINK_REG are used for enabling the DMA Engine and are self-cleared by hardware. When I2S_OUTLINK_START is set to 1, the DMA Engine starts processing the outbound linked-list descriptor and gets prepared to send data. When I2S_INLINK_START is set to 1, the DMA Engine starts processing the inbound linked-list descriptor and gets prepared to receive data.

Software should configure the I2S DMA as follows:

1. Configure I2S-controller-related registers;

2. Reset the DMA state machine and FIFO parameters;
3. Configure DMA-related registers for operation;
4. In I2S master mode, set I2S_TX_START bit or I2S_RX_START bit to initiate an I2S operation;
   In I2S slave mode, set I2S_TX_START bit or I2S_RX_START bit and wait for data transfer to be initiated by the host device.

For more information on I2S DMA interrupts, please see Section DMA Interrupts, in Chapter I2S.

7. SPI

7.1 Overview

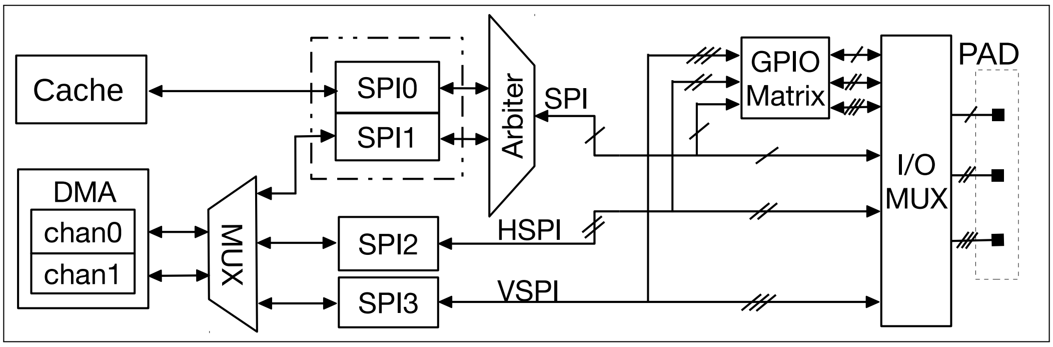


Figure 16: SPI Architecture

As Figure 16 shows, ESP32 integrates four SPI controllers which can be used to communicate with external devices that use the SPI protocol. Controller SPI0 is used as a buffer for accessing external memory. Controller SPI1 can be used as a master. Controllers SPI2 and SPI3 can be configured as either a master or a slave. When used as a master, each SPI controller can drive multiple CS signals (CS0 ~ CS2) to activate multiple slaves. Controllers SPI1 ~ SPI3 share two DMA channels.

The SPI signal buses consist of D, Q, CS0-CS2, CLK, WP, and HD signals, as Table 25 shows. Controllers SPI0 and SPI1 share one signal bus through an arbiter; the signals of the shared bus start with ”SPI”. Controllers SPI2 and SPI3 use signal buses starting with ”HSPI” and ”VSPI” respectively. The I/O lines included in the above-mentioned signal buses can be mapped to pins via either the IO_MUX module or the GPIO matrix. (Please refer to Chapter IO_MUX for details.)

The SPI controller supports four-line half-duplex and full-duplex communication (MOSI, MISO, CS, and CLK lines) and three-line-bit half-duplex-only communication (DATA, CS, and CLK lines) in GP-SPI mode. In QSPI mode, a SPI controller accesses the flash or SRAM by using signal buses D, Q, CS0 ~ CS2, CLK, WP, and HD as a four-bit parallel SPI bus. The mapping between the GP-SPI signal bus and the QSPI signal bus is shown in Table 25.

Table 25: SPI Signal and Pin Signal Function Mapping

Four-line GP-SPI Full-duplex signal bus	Three-line GP-SPI Half-duplex signal bus	QSPI Signal bus	Pin function signals		
			SPI signal bus	HSPI signal bus	VSPI signal bus
MOSI	DATA	D	SPID	HSPID	VSPID
MISO	-	Q	SPIQ	HSPIQ	VSPIQ
CS	CS	CS	SPICS0	HSPICS0	VSPICS0
CLK	CLK	CLK	SPICLK	HSPICLK	VSPICLK
-	-	WP	SPIWP	HSPIWP	VSPIWP
-	-	HD	SPIHD	HSPIHD	VSPIHD

7.2 SPI Features

General Purpose SPI (GP-SPI)

- Programmable data transaction length, in multiples of 1 byte
- Four-line full-duplex communication and three-line half-duplex communication support
- Master mode and slave mode
- Programmable CPOL and CPHA
- Programmable clock

Parallel QSPI

- Communication format support for specific slave devices such as flash
- Programmable communication format
- Six variations of flash-read operations available
- Automatic shift between flash and SRAM access
- Automatic wait states for flash access

SPI DMA Support

- Support for sending and receiving data using linked lists

SPI Interrupt Hardware

- SPI interrupts
- SPI DMA interrupts

7.3 GP-SPI

The SPI1 ~ SPI3 controllers can communicate with other slaves as a standard SPI master. Every SPI master can be connected to three slaves at most by default. In non-DMA mode, the maximum length of data received/sent in one burst is 64 bytes. The data length is in multiples of 1 byte.

7.3.1 GP-SPI Master Mode

The SPI master mode supports four-line full-duplex communication and three-line half-duplex communication. The connections needed for four-line full-duplex communications are outlined in Figure 17.

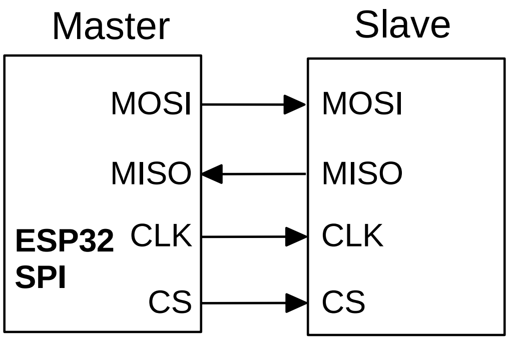


Figure 17: SPI Master and Slave Full-duplex Communication

For four-line full-duplex communication, the length of received and sent data needs to be set by configuring the SPI_MISO_DLEN_REG, SPI_MOSI_DLEN_REG registers for master mode as well as

SPI_SLV_RDBUF_DLEN_REG, SPI_SLV_WRBUF_DLEN_REG registers for slave mode. The SPI_DOUTDIN bit and SPI_USR_MOSI bit in register SPI_USER_REG should also be configured. The SPI_USR bit in register SPI_CMD_REG needs to be configured to initialize data transfer.

If ESP32 SPI is configured as a slave using three-line half-duplex communication, the master-slave communication should meet a certain communication format. Please refer to 7.3.2.1 for details. For example, if ESP32 SPI acts as a slave, the communication format should be: command + address + received/sent data. The address length of the master should be the same as that of the slave; the value of the address should be 0.

Note:
When using ESP32 as a master in half-duplex communication, the communication format "command + address + sent data + received data" and "sent data + received data" are not applicable to DMA.

The byte order in which ESP32 SPI reads and writes is controlled by the SPI_RD_BYTE_ORDER bit and the SPI_WR_BYTE_ORDER bit in register SPI_USER_REG. The bit order is controlled by the SPI_RD_BIT_ORDER bit and the SPI_WR_BIT_ORDER bit in register SPI_CTRL_REG.

7.3.2 GP-SPI Slave Mode

ESP32 SPI2 ~ SPI3 can communicate with other host devices as a slave device. ESP32 SPI should use particular protocols when acting as a slave. Data received or sent at one time can be no more than 64 bytes when not using DMA. During a valid read/write process, the appropriate CS signal must be maintained at a low level. If the CS signal is pulled up during transmission, the internal state of the slave will be reset.

7.3.2.1 Communication Format Supported by GP-SPI Slave

The communication format of ESP32 SPI is: command + address + read/write data. When using half-duplex communication, the slave read and write operations use fixed hardware commands from which the address part can not be removed. The command is specified as follows:

1. command: length: 3 ~ 16 bits; Master Out Slave In (MOSI).
2. address: length: 1 ~ 32 bits; Master Out Slave In (MOSI).
3. data read/write: length 0 ~ 512 bits (64 bytes); Master Out Slave In (MOSI) or Master In Slave Out (MISO).

When ESP32 SPI is used as a slave in full-duplex communication, data transaction can be directly initiated without the master sending command and address. However, please note that the CS should be pulled low at least one SPI clock period before a read/write process is initiated, and should be pulled high at least one SPI clock period after the read/write process is completed.

7.3.2.2 Command Definitions Supported by GP-SPI Slave in Half-duplex Mode

The minimum length of a command received by the slave should be three bits. The lowest three bits correspond to fixed hardware read and write operations as follows:

1. 0x1 (received by slave): Writes data sent by the master into the slave status register via MOSI.
2. 0x2 (received by slave): Writes data sent by the master into the slave data buffer.
3. 0x3 (sent by slave): Sends data in the slave buffer to master via MISO.

4. 0x4 (sent by slave): Sends data in the slave status register to master via MISO.
5. 0x6 (received and then sent by slave): Writes master data on MOSI into data buffer and then sends the date in the slave data buffer to MISO.

The master can write the slave status register SPI_SLV_WR_STATUS_REG, and decide whether to read data from register SPI_SLV_WR_STATUS_REG or register SPI_RD_STATUS_REG via the SPI_SLV_STATUS_READBACK bit in the register SPI_SLAVE1_REG. The SPI master can maintain communication with the slave by reading and writing slave status register, thus realizing relatively complex communication with ease.

7.3.3 GP-SPI Data Buffer

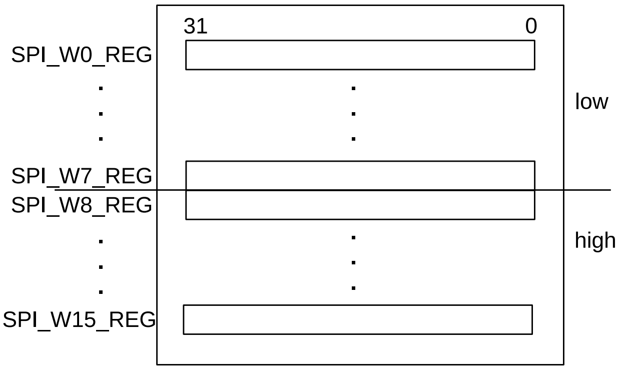


Figure 18: SPI Data Buffer

ESP32 SPI has 16 x 32 bits of data buffer to buffer data-send and data-receive operations. As is shown in Figure 18, received data is written from the low byte of SPI_W0_REG by default and the writing ends with SPI_W15_REG. If the data length is over 64 bytes, the extra part will be written from SPI_W0_REG.

Data buffer blocks SPI_W0_REG ~ SPI_W7_REG and SPI_W8_REG ~ SPI_W15_REG data correspond to the lower part and the higher part respectively. They can be used separately, and are controlled by the SPI_USR_MOSI_HIGHPART bit and the SPI_USR_MISO_HIGHPART bit in register SPI_USER_REG. For example, if SPI is configured as a master, when SPI_USR_MOSI_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for sending data; when SPI_USR_MISO_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for receiving data. If SPI acts as a slave, when SPI_USR_MOSI_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for receiving data; when SPI_USR_MISO_HIGHPART = 1, SPI_W8_REG ~ SPI_W15_REG are used as buffer for sending data.

7.4 GP-SPI Clock Control

The maximum output clock frequency of ESP32 GP-SPI master is $f_{\text{apb}}/2$, and the maximum input clock frequency of the ESP32 GP-SPI slave is $f_{\text{apb}}/8$. The master can derive other clock frequencies via frequency division.

$$f_{\text{spi}} = \frac{f_{\text{apb}}}{\text{(SPI\_CLKCNT\_N+1)(SPI\_CLKDIV\_PRE+1)}}$$

SPI_CLKCNT_N and SPI_CLKDIV_PRE are two bits of register SPI_CLOCK_REG (Please refer to 7.8 Register Description for details). When the SPI_CLK_EQU_SYSCLK bit in the register SPI_CLOCK_REG is set to 1, and the other bits are set to 0, SPI output clock frequency is f_{apb}. For other clock frequencies, SPI_CLK_EQU_SYSCLK needs to be 0.

7.4.1 GP-SPI Clock Polarity (CPOL) and Clock Phase (CPHA)

The clock polarity and clock phase of ESP32 SPI are controlled by the SPI_CK_IDLE_EDGE bit in register SPI_PIN_REG, the SPI_CK_OUT_EDGE bit and the SPI_CK_I_EDGE bit in register SPI_USER_REG, the SPI_MISO_DELAY_MODE[1:0] bit, the SPI_MISO_DELAY_NUM[2:0] bit, the SPI_MOSI_DELAY_MODE[1:0] bit, and the SPI_MOSI_DELAY_MUM[2:0] bit in register SPI_CTRL2_REG. Table 26 and Table 27 show the clock polarity and phase as well as the corresponding register values for ESP32 SPI master and slave, respectively.

Table 26: Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Master

Registers	mode0	mode1	mode2	mode3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0
SPI_MISO_DELAY_MODE	2(0)	1(0)	1(0)	2(0)
SPI_MISO_DELAY_NUM	0	0	0	0
SPI_MOSI_DELAY_MODE	0	0	0	0
SPI_MOSI_DELAY_NUM	0	0	0	0

Table 27: Clock Polarity and Phase, and Corresponding SPI Register Values for SPI Slave

Registers	mode0	mode1	mode2	mode3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_I_EDGE	0	1	1	0
SPI_MISO_DELAY_MODE	0	0	0	0
SPI_MISO_DELAY_NUM	0	0	0	0
SPI_MOSI_DELAY_MODE	2	1	1	2
SPI_MOSI_DELAY_NUM	0	0	0	0

1. mode0 means CPOL=0, CPHA=0. When SPI is idle, the clock output is logic low; data change on the falling edge of the SPI clock and are sampled on the rising edge;
2. mode1 means CPOL=0, CPHA=1. When SPI is idle, the clock output is logic low; data change on the rising edge of the SPI clock and are sampled on the falling edge;
3. mode2 means when CPOL=1, CPHA=0. When SPI is idle, the clock output is logic high; data change on the rising edge of the SPI clock and are sampled on the falling edge;
4. mode3 means when CPOL=1, CPHA=1. When SPI is idle, the clock output is logic high; data change on the falling edge of the SPI clock and are sampled on the rising edge.

7.4.2 GP-SPI Timing

The data signals of ESP32 GP-SPI can be mapped to physical pins via IO_MUX or via IO_MUX and GPIO matrix. When signals pass through the matrix, they will be delayed by two clk_{apb} clock cycles.

When GP-SPI is used as master and the data signals are not received by the SPI controller via GPIO matrix, if GP-SPI output clock frequency is not higher than $clk_{\text{apb}}/2$, register SPI_MISO_DELAY_MODE should be set to 0 when configuring the clock polarity. If GP-SPI output clock frequency is not higher than $clk_{\text{apb}}/4$, register

SPI_MISO_DELAY_MODE can be set to the corresponding value in Table 26 when configuring the clock polarity.

When GP-SPI is used in master mode and the data signals enter the SPI controller via the GPIO matrix:

1. If GP-SPI output clock frequency is $clk_{\text{apb}}/2$, register SPI_MISO_DELAY_MODE should be set to 0 and the dummy state should be enabled (SPI_USR_DUMMY = 1) for one clk_{spi} clock cycle (SPI_USR_DUMMY_CYCLELEN = 0) when configuring the clock polarity;
2. If GP-SPI output clock frequency is $clk_{\text{apb}}/4$, register SPI_MISO_DELAY_MODE should be set to 0 when configuring the clock polarity;
3. If GP-SPI output clock frequency is not higher than $clk_{\text{apb}}/8$, register SPI_MISO_DELAY_MODE can be set to the corresponding value in Table 26 when configuring the clock polarity.

When GP-SPI is used in slave mode, the maximum slave input clock frequency is $f_{\text{apb}}/8$. In addition, the clock signal and the data signals should be routed to the SPI controller via the same path, i.e., neither the clock signal nor the data signals enter the SPI controller via the GPIO matrix, or both the clock signal and the data signals enter the SPI controller via the GPIO matrix. This is important in ensuring that the signals are not delayed by different time periods before they reach the SPI hardware.

7.5 Parallel QSPI

ESP32 SPI controllers support SPI bus memory devices (such as flash and SRAM). The hardware connection between the SPI pins and the memories is shown by Figure 19.

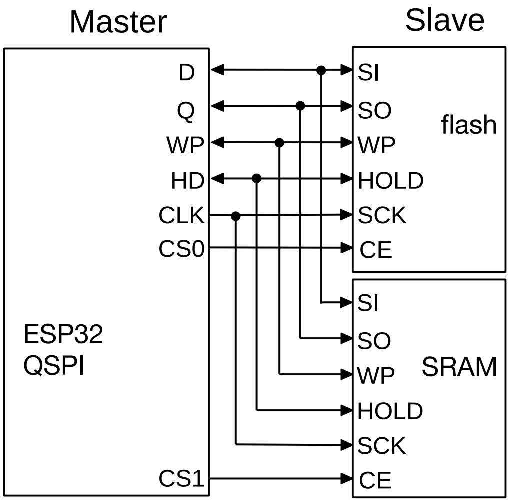


Figure 19: Parallel QSPI

SPI1, SPI2 and SPI3 controllers can also be configured as QSPI master to connect to external memory. The maximum output clock frequency of the SPI memory interface is f_{apb}, with the same clock configuration as that of the GP-SPI master.

ESP32 QSPI supports flash-read operation in one-line mode, two-line mode, and four-line mode.

7.5.1 Communication Format of Parallel QSPI

To support communication with special slave devices, ESP32 QSPI implements a specifically designed communication protocol. The communication format of ESP32 QSPI master is command + address + read/write data, as shown in Figure 20, with details as follows:

1. Command: length: 1 ~ 16 bits; Master Out Slave In.
2. Address: length: 0 ~ 64 bits; Master Out Slave In.
3. Data read/write: length: 0 ~ 512 bits (64 bytes); Master Out Slave In or Master In Slave Out.

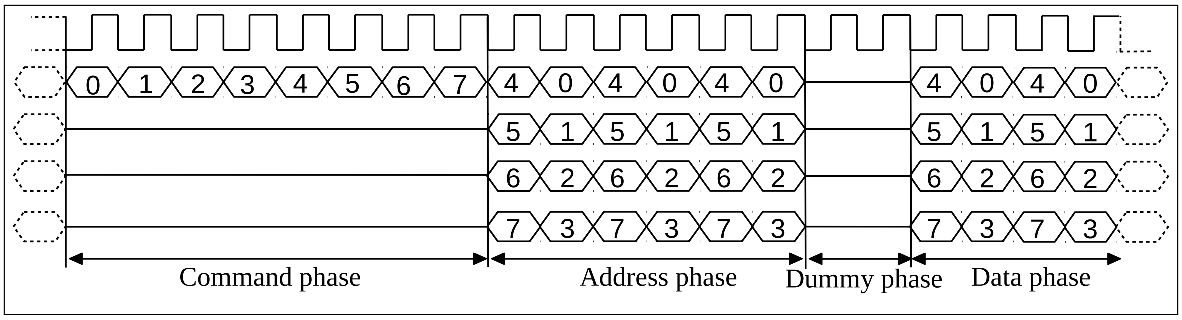


Figure 20: Communication Format of Parallel QSPI

When ESP32 SPI is configured as a master and communicates with slaves that use the SPI protocol, options such as command, address, data, etc., can be adjusted as required by the specific application. When ESP32 SPI reads special devices such as Flash and SRAM, a dummy state with a programmable length can be inserted between the address phase and the data phase.

7.6 GP-SPI Interrupt Hardware

ESP32 SPI generates two types of interrupts. One is the SPI interrupt and the other is the SPI DMA interrupt.

ESP32 SPI reckons the completion of send and/or receive operations as the completion of one operation from the controller and generates one interrupt. When ESP32 SPI is configured to slave mode, the slave will generate read/write status registers and read/write buffer data interrupts according to different operations.

7.6.1 SPI Interrupts

The SPI_*_INTEN bits in the SPI_SLAVE_REG register can be set to enable SPI interrupts. When an SPI interrupt happens, the interrupt flag in the corresponding SPI_*_DONE register will get set. This flag is writable, and an interrupt can be cleared by setting the bit to zero.

- SPI_TRANS_DONE_INT: Triggered when a SPI operation is done.
- SPI_SLV_WR_STA_INT: Triggered when a SPI slave status write is done.
- SPI_SLV_RD_STA_INT: Triggered when a SPI slave status read is done.
- SPI_SLV_WR_BUF_INT: Triggered when a SPI slave buffer write is done.
- SPI_SLV_RD_BUD_INT: Triggered when a SPI slave buffer read is done.

7.6.2 DMA Interrupts

- SPI_OUT_TOTAL_EOF_INT: Triggered when all linked lists are sent.
- SPI_OUT_EOF_INT: Triggered when one linked list is sent.
- SPI_OUT_DONE_INT: Triggered when the last linked list item has zero length.
- SPI_IN_SUC_EOF_INT: Triggered when all linked lists are received.
- SPI_IN_ERR_EOF_INT: Triggered when there is an error receiving linked lists.
- SPI_IN_DONE_INT: Triggered when the last received linked list had a length of 0.
- SPI_INLINK_DSCR_ERROR_INT: Triggered when the received linked list is invalid.
- SPI_OUTLINK_DSCR_ERROR_INT: Triggered when the linked list to be sent is invalid.
- SPI_INLINK_DSCR_EMPTY_INT: Triggered when no valid linked list is available.

7.7 Register Summary

Name	Description	SPI0	SPI1	SPI2	SPI3	Acc
Control and configuration registers						
SPI_CTRL_REG	Bit order and QIO/DIO/QOUT/DOUT mode settings	3FF43008	3FF42008	3FF65000	3FF65000	R/W
SPI_CTRL1_REG	CS delay configuration	3FF4300C	3FF4200C	3FF6400C	3FF6400C	R/W
SPI_CTRL2_REG	Timing configuration	3FF43014	3FF42014	3FF64014	3FF64014	R/W
SPI_CLOCK_REG	Clock configuration	3FF43018	3FF42018	3FF64018	3FF64018	R/W
SPI_PIN_REG	Polarity and CS configuration	3FF43034	3FF42034	3FF64034	3FF64034	R/W
Slave mode configuration registers						
SPI_SLAVE_REG	Slave mode configuration and interrupt status	3FF43038	3FF42038	3FF64038	3FF64038	R/W
SPI_SLAVE1_REG	Slave data bit lengths	3FF4303C	3FF4203C	3FF6403C	3FF6403C	R/W
SPI_SLAVE2_REG	Dummy cycle length configuration	3FF43040	3FF42040	3FF64040	3FF64040	R/W
SPI_SLAVE3_REG	Read/write status/buffer register	3FF43044	3FF42044	3FF64044	3FF64044	R/W
SPI_SLV_WR_STATUS_REG	Slave status/higher master address	3FF43030	3FF42030	3FF64030	3FF64030	R/W
SPI_SLV_WRBUF_DLEN_REG	Write-buffer operation length	3FF43048	3FF42048	3FF64048	3FF64048	R/W
SPI_SLV_RDBUF_DLEN_REG	Read-buffer operation length	3FF4304C	3FF4204C	3FF6404C	3FF6404C	R/W
SPI_SLV_RD_BIT_REG	Read data operation length	3FF43064	3FF42064	3FF64064	3FF64064	R/W

User-defined command mode registers						
SPI_CMD_REG	Start user-defined command	3FF43000	3FF42000	3FF64000	3FF64000	R/W
SPI_ADDR_REG	Address data	3FF43004	3FF42004	3FF64004	3FF64004	R/W
SPI_USER_REG	User defined command configuration	3FF4301C	3FF4201C	3FF6401C	3FF6401C	R/W
SPI_USER1_REG	Address and dummy cycle configuration	3FF43020	3FF42020	3FF64020	3FF64020	R/W
SPI_USER2_REG	Command length and value configuration	3FF43024	3FF42024	3FF64024	3FF64024	R/W
SPI_MOSI_DLEN_REG	MOSI length	3FF43028	3FF42028	3FF64028	3FF64028	R/W
SPI_W0_REG	SPI data register 0	3FF43080	3FF42080	3FF64080	3FF64080	R/W
SPI_W1_REG	SPI data register 1	3FF43084	3FF42084	3FF64084	3FF64084	R/W
SPI_W2_REG	SPI data register 2	3FF43088	3FF42088	3FF64088	3FF64088	R/W
SPI_W3_REG	SPI data register 3	3FF4308C	3FF4208C	3FF6408C	3FF6408C	R/W
SPI_W4_REG	SPI data register 4	3FF43090	3FF42090	3FF64090	3FF64090	R/W
SPI_W5_REG	SPI data register 5	3FF43094	3FF42094	3FF64094	3FF64094	R/W
SPI_W6_REG	SPI data register 6	3FF43098	3FF42098	3FF64098	3FF64098	R/W
SPI_W7_REG	SPI data register 7	3FF4309C	3FF4209C	3FF6409C	3FF6409C	R/W
SPI_W8_REG	SPI data register 8	3FF430A0	3FF420A0	3FF640A0	3FF640A0	R/W
SPI_W9_REG	SPI data register 9	3FF430A4	3FF420A4	3FF640A4	3FF640A4	R/W
SPI_W10_REG	SPI data register 10	3FF430A8	3FF420A8	3FF640A8	3FF640A8	R/W
SPI_W11_REG	SPI data register 11	3FF430AC	3FF420AC	3FF640AC	3FF640AC	R/W
SPI_W12_REG	SPI data register 12	3FF430B0	3FF420B0	3FF640B0	3FF640B0	R/W
SPI_W13_REG	SPI data register 13	3FF430B4	3FF420B4	3FF640B4	3FF640B4	R/W
SPI_W14_REG	SPI data register 14	3FF430B8	3FF420B8	3FF640B8	3FF640B8	R/W
SPI_W15_REG	SPI data register 15	3FF430BC	3FF420BC	3FF640BC	3FF640BC	R/W
SPI_TX_CRC_REG	CRC32 of 256 bits of data (SPI1 only)	3FF430C0	3FF420C0	3FF640C0	3FF640C0	R/W
Status registers						
SPI_RD_STATUS_REG	Slave status and fast read mode	3FF43010	3FF42010	3FF64010	3FF64010	R/W
DMA configuration registers						
SPI_DMA_CONF_REG	DMA configuration register	3FF43100	3FF42100	3FF64100	3FF64100	R/W
SPI_DMA_OUT_LINK_REG	DMA outlink address and configuration	3FF43104	3FF42104	3FF64104	3FF64104	R/W
SPI_DMA_IN_LINK_REG	DMA inlink address and configuration	3FF43108	3FF42108	3FF64108	3FF64108	R/W
SPI_DMA_STATUS_REG	DMA status	3FF4310C	3FF4210C	3FF6410C	3FF6410C	RO
SPI_IN_ERR_EOF_DES_ADDR_REG	Descriptor address where an error occurs	3FF43120	3FF42120	3FF64120	3FF64120	RO

SPI_IN_SUC_EOF_DES_ADDR_REG	Descriptor address where EOF occurs	3FF43124	3FF42124	3FF64124	3FF64124	RO
SPI_INLINK_DSCR_REG	Current descriptor pointer	3FF43128	3FF42128	3FF64128	3FF64128	RO
SPI_INLINK_DSCR_BF0_REG	Next descriptor data pointer	3FF4312C	3FF4212C	3FF6412C	3FF6412C	RO
SPI_INLINK_DSCR_BF1_REG	Current descriptor data pointer	3FF43130	3FF42130	3FF64130	3FF64130	RO
SPI_OUT_EOF_BFR_DES_ADDR_REG	Relative buffer address where EOF occurs	3FF43134	3FF42134	3FF64134	3FF64134	RO
SPI_OUT_EOF_DES_ADDR_REG	Descriptor address where EOF occurs	3FF43138	3FF42138	3FF64138	3FF64138	RO
SPI_OUTLINK_DSCR_REG	Current descriptor pointer	3FF4313C	3FF4213C	3FF6413C	3FF6413C	RO
SPI_OUTLINK_DSCR_BF0_REG	Next descriptor data pointer	3FF43140	3FF42140	3FF64140	3FF64140	RO
SPI_OUTLINK_DSCR_BF1_REG	Current descriptor data pointer	3FF43144	3FF42144	3FF64144	3FF64144	RO
SPI_DMA_RSTATUS_REG	DMA memory read status	3FF43148	3FF42148	3FF64148	3FF64148	RO
SPI_DMA_TSTATUS_REG	DMA memory write status	3FF4314C	3FF4214C	3FF6414C	3FF6414C	RO
DMA interrupt registers						
SPI_DMA_INT_RAW_REG	Raw interrupt status	3FF43114	3FF42114	3FF64114	3FF64114	RO
SPI_DMA_INT_ST_REG	Masked interrupt status	3FF43118	3FF42118	3FF64118	3FF64118	RO
SPI_DMA_INT_ENA_REG	Interrupt enable bits	3FF43110	3FF42110	3FF64110	3FF64110	R/W
SPI_DMA_INT_CLR_REG	Interrupt clear bits	3FF4311C	3FF4211C	3FF6411C	3FF6411C	R/W

7.8 Registers

Register 7.1: SPI_CMD_REG (0x0)

(reserved)	SPI_USR	(reserved)	
31 19	18	35 18	
0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_USR This bit is used to enable user-defined commands. An operation will be triggered when this bit is set. The bit will be cleared once the operation is done. (R/W)

Register 7.2: SPI_ADDR_REG (0x4)

31 0	
0x000000000	Reset

SPI_ADDR_REG Address to slave or from master. If the address length is bigger than 32 bits, SPI_SLV_WR_STATUS_REG contains the lower 32 bits while this register contains the higher address bits. (R/W)

Register 7.3: SPI_CTRL_REG (0x8)

(reserved)	SPI_WR_BIT_ORDER	SPI_RD_BIT_ORDER	SPI_FREAD_QIO	SPI_FREAD_DIO	(reserved)	SPI_WP	SPI_FREAD_QUAD	(reserved)	SPI_FREAD_DUAL	SPI_FASTRD_MODE	(reserved)	
31 27	26	25	24	23	22	21	20	19 15	14	13	25 13	
0 0 0 0 0	0	0	0	0	0	1	0	0 0 0 0 0	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_WR_BIT_ORDER This bit determines the bit order for command, address and MOSI data writes. 1: sends LSB first; 0: sends MSB first. (R/W)

SPI_RD_BIT_ORDER This bit determines the bit order for MOSI data reads. 1: receives LSB first; 0: receives MSB first. (R/W)

SPI_FREAD_QIO This bit determines whether to use four data lines for address writes and MOSI data reads or not. 1: enable; 0: disable. (R/W)

SPI_FREAD_DIO This bit determines whether to use two data lines for address writes and MOSI data reads or not. 1: enable; 0: disable. (R/W)

SPI_WP This bit determines the write-protection signal output when SPI is idle. 1: output high; 0: output low. (R/W)

SPI_FREAD_QUAD This bit determines whether to use four data lines for MOSI data reads or not. 1: enable; 0: disable. (R/W)

SPI_FREAD_DUAL This bit determines whether to use two data lines for MOSI data reads or not. 1: enable; 0: disable. (R/W)

SPI_FASTRD_MODE This bit is used to enable spi_fread_qio, spi_fread_dio, spi_fread_qout, and spi_fread_dout. 1: enable 0: disable. (R/W)

Register 7.4: SPI_CTRL1_REG (0xC)

SPI_CS_HOLD_DELAY	(reserved)	
31 28	55 28	
0x05	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_CS_HOLD_DELAY The number of SPI clock cycles by which the SPI CS signal is delayed. (R/W)

Register 7.5: SPI_RD_STATUS_REG (0x10)

SPI_STATUS_EXT		SPI_STATUS	
31 24	23 16	15 0	
0x000	0x000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_STATUS_EXT In slave mode, this is the status for the master to read. (R/W)

SPI_STATUS In slave mode, this is the status for the master to read. (R/W)

Register 7.6: SPI_CTRL2_REG (0x14)

SPI_CS_DELAY_NUM	SPI_CS_DELAY_MODE	SPI_MOSI_DELAY_NUM	SPI_MOSI_DELAY_MODE	SPI_MISO_DELAY_NUM	SPI_MISO_DELAY_MODE	SPI_CK_OUT_HIGH_MODE	reserved	SPI_HOLD_TIME	SPI_SETUP_TIME	
31 28	27 26	25 23	22 21	20 18	17 16	15 12	11 8	7 4	3 0	
0x00	0x0	0x0	0x0	0x0	0x0	0x00	0x00	0x01	0x01	Reset

SPI_CS_DELAY_NUM The spi_cs signal is delayed by the number of system clock cycles configured here. (R/W)

SPI_CS_DELAY_MODE This register field determines the way the spi_cs signal is delayed by spi_clk. (R/W)

0: none.

1: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, spi_cs is delayed by half a cycle, otherwise it is delayed by one cycle.

2: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, spi_cs is delayed by one cycle, otherwise it is delayed by half a cycle.

3: the spi_cs signal is delayed by one cycle.

SPI_MOSI_DELAY_NUM The MOSI signals are delayed by the number of system clock cycles configured here. (R/W)

SPI_MOSI_DELAY_MODE This register field determines the way the MOSI signals are delayed by spi_clk. (R/W)

0: none.

1: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MOSI signals are delayed by half a cycle, otherwise they are delayed by one cycle.

2: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MOSI signals are delayed by one cycle, otherwise they are delayed by half a cycle.

3: the MOSI signals are delayed one cycle.

SPI_MISO_DELAY_NUM The MISO signals are delayed by the number of system clock cycles specified here. (R/W)

SPI_MISO_DELAY_MODE This register field determines the way MISO signals are delayed by spi_clk. (R/W)

0: none.

1: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MISO signals are delayed by half a cycle, otherwise they are delayed by one cycle.

2: if SPI_CK_OUT_EDGE or SPI_CK_I_EDGE is set, the MISO signals are delayed by one cycle, otherwise they are delayed by half a cycle.

3: the MISO signals are delayed by one cycle.

SPI_HOLD_TIME The number of spi_clk cycles by which CS pin signals are delayed. These bits are used in conjunction with the SPI_CS_HOLD bit. (R/W)

SPI_SETUP_TIME The number of spi_clk cycles for which spi_cs is made active before the SPI data transaction starts. This register field is used when SPI_CS_SETUP is set. (R/W)

Register 7.7: SPI_CLOCK_REG (0x18)

SPI_CLK_EQU_SYSCLK	SPI_CLKDIV_PRE	SPI_CLKCNT_N	SPI_CLKCNT_H	SPI_CLKCNT_L	
31	30 18	17 12	11 6	5 0	
1	0 0 0 0 0 0 0 0 0 0 0 0 0	0x03	0x01	0x03	Reset

SPI_CLK_EQU_SYSCLK In master mode, when this bit is set to 1, spi_clk is equal to system clock; when set to 0, spi_clk is divided from system clock. (R/W)

SPI_CLKDIV_PRE In master mode, the value of this register field is the pre-divider value for spi_clk, minus one. (R/W)

SPI_CLKCNT_N In master mode, this is the divider for spi_clk minus one. The spi_clk frequency is system_clock/(SPI_CLKDIV_PRE+1)/(SPI_CLKCNT_N+1). (R/W)

SPI_CLKCNT_H For a 50% duty cycle, set this to floor((SPI_CLKCNT_N+1)/2-1). (R/W)

SPI_CLKCNT_L In master mode, this must be equal to SPI_CLKCNT_N. In slave mode this must be 0. (R/W)

Register 7.8: SPI_USER_REG (0x1C)

SPI_USR_COMMAND	SPI_USR_ADDR	SPI_USR_DUMMY	SPI_USR_MISO	SPI_USR_MOSI	SPI_USR_DUMMY_IDLE	SPI_USR_MOSI_HIGHPART	SPI_USR_MISO_HIGHPART	(reserved)	SPI_SIO	SPI_FWRITE_QIO	SPI_FWRITE_DIO	SPI_FWRITE_QUAD	SPI_FWRITE_DUAL	SPI_WR_BYTE_ORDER	SPI_RD_BYTE_ORDER	(reserved)	SPI_CK_OUT_EDGE	SPI_CK_I_EDGE	SPI_CS_SETUP	SPI_CS_HOLD	(reserved)	SPI_DOUTDIN	
31	30	29	28	27	26	25	24	23 17	16	15	14	13	12	11	10	9 8	7	6	5	4	3 1	0	
1	0	0	0	0	0	0	0	0 0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	1	0	0	0 0 0	0	Reset

SPI_USR_COMMAND This bit enables the command phase of an operation. (R/W)

SPI_USR_ADDR This bit enables the address phase of an operation. (R/W)

SPI_USR_DUMMY This bit enables the dummy phase of an operation. (R/W)

SPI_USR_MISO This bit enables the read-data phase of an operation. (R/W)

SPI_USR_MOSI This bit enables the write-data phase of an operation. (R/W)

SPI_USR_DUMMY_IDLE The spi_clk signal is disabled in the dummy phase when the bit is set. (R/W)

SPI_USR_MOSI_HIGHPART If set, data written to the device is only read from SPI_W8-SPI_W15 of the SPI buffer. (R/W)

SPI_USR_MISO_HIGHPART If set, data read from the device is only written to SPI_W8-SPI_W15 of the SPI buffer. (R/W)

SPI_SIO Set this bit to enable three-line half-duplex communication where MOSI and MISO signals share the same pin. (R/W)

SPI_FWRITE_QIO This bit enables the use of four data lines for address and MISO data writes. 1: enable; 0: disable. (R/W)

SPI_FWRITE_DIO This bit enables the use of two data lines for address and MISO data writes. 1: enable; 0: disable. (R/W)

SPI_FWRITE_QUAD This bit enables the use of four data lines for MISO data writes. 1: enable; 0: disable. (R/W)

SPI_FWRITE_DUAL This bit determines whether to use two data lines for MISO data writes or not. 1: enable; 0: disable. (R/W)

SPI_WR_BYTE_ORDER This bit determines the byte-endianness for writing command, address, and MOSI data. 1: big-endian; 0: litte-endian. (R/W)

SPI_RD_BYTE_ORDER This bit determines the byte-endianness for reading MISO data. 1: big-endian; 0: little_endian. (R/W)

SPI_CK_OUT_EDGE This bit, combined with SPI_MOSI_DELAY_MODE, sets the MOSI signal delay mode. (R/W)

SPI_CK_I_EDGE In slave mode, the bit is the same as SPI_CK_OUT_EDGE in master mode. It is combined with SPI_MISO_DELAY_MODE. (R/W)

SPI_CS_SETUP Setting this bit enables a delay between spi_cs being active and starting data transfer, as specified in SPI_SETUP_TIME. This bit only is valid in half-duplex mode, that is, when SPI_DOUTDIN is not set. (R/W)

SPI_CS_HOLD Setting this bit enables a delay between the end of a transmission and spi_cs being made inactive, as specified in SPI_HOLD_TIME. (R/W)

SPI_DOUTDIN Set the bit to enable full-duplex communication, meaning that MOSI data is sent out at the same time MISO data is received. 1: enable; 0: disable. (R/W)

Register 7.9: SPI_USER1_REG (0x20)

SPI_USR_ADDR_BITLEN	(reserved)	SPI_USR_DUMMY_CYCLELEN	
31 26	25 8	7 0	
23	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	7	Reset

SPI_USR_ADDR_BITLEN The bit length of the address phase minus one. (RO)

SPI_USR_DUMMY_CYCLELEN The number of spi_clk cycles for the dummy phase, minus one. (R/W)

Register 7.10: SPI_USER2_REG (0x24)

SPI_USR_COMMAND_BITLEN	(reserved)	SPI_USR_COMMAND_VALUE	
31 28	27 16	15 0	
7	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_USR_COMMAND_BITLEN The bit length of the command phase minus one. (R/W)

SPI_USR_COMMAND_VALUE The value of the command. (R/W)

Register 7.11: SPI_MOSI_DLEN_REG (0x28)

(reserved)	SPI_USR_MOSI_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_USR_MOSI_DBITLEN The bit length of the data to be written to the device minus one. (R/W)

Register 7.12: SPI_MISO_DLEN_REG (0x2C)

(reserved)	SPI_USR_MISO_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_USR_MISO_DBITLEN The bit length of the data to be read from the device, minus one. (R/W)

Register 7.13: SPI_SLV_WR_STATUS_REG (0x30)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_SLV_WR_STATUS_REG In the slave mode this register is the status register for the master to write into. In the master mode, if the address length is bigger than 32 bits, this register contains the lower 32 bits. (R/W)

Register 7.14: SPI_PIN_REG (0x34)

(reserved)	SPI_CS_KEEP_ACTIVE	SPI_CK_IDLE_EDGE	(reserved)	SPI_MASTER_CK_SEL	(reserved)	SPI_MASTER_CS_POL	SPI_CK_DIS	(reserved)	SPI_CS2_DIS	SPI_CS1_DIS	SPI_CS0_DIS	
31	30	29	28 14	13 11	10 9	8 6	5	4 3	2	1	0	
0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	0 0	0 0 0 0 0	0	0 0	1	1	0	Reset

SPI_CS_KEEP_ACTIVE When set, the spi_cs will be kept active even when not in a data transaction. (R/W)

SPI_CK_IDLE_EDGE The idle state of the spi_clk line. (R/W)
1: the spi_clk line is high when idle;
0: the spi_clk line is low when idle.

SPI_MASTER_CK_SEL This register field contains one bit per spi_cs line. When a bit is set in master mode, the corresponding spi_cs line is made active and the spi_cs pin outputs spi_clk. (R/W)

SPI_MASTER_CS_POL This register filed selects the polarity of the spi_cs line. It contains one bit per spi_cs line. Possible values of the bits: (R/W)
0: spi_cs is active-low;
1: spi_cs is active-high.

SPI_CK_DIS When set, output of the spi_clk signal is disabled. (R/W)

SPI_CS2_DIS This bit enables the SPI CS2 pin. 1: disables CS2; 0: spi_cs2 is active during the data transaction. (R/W)

SPI_CS1_DIS This bit enables the SPI CS1 pin. 1: disables CS1; 0: spi_cs1 is active during the data transaction (R/W)

SPI_CS0_DIS This bit enables the SPI CS0 pin. 1: disables CS0; 0: spi_cs0 is active during the data transaction. (R/W)

Register 7.15: SPI_SLAVE_REG (0x38)

SPI_SYNC_RESET	SPI_SLAVE_MODE	SPI_SLV_WR_RD_BUF_EN	SPI_SLV_WR_RD_STA_EN	SPI_SLV_CMD_DEFINE	SPI_TRANS_CNT	SPI_SLV_LAST_STATE	SPI_SLV_LAST_COMMAND	(reserved)	SPI_CS_I_MODE	SPI_TRANS_INTEN	SPI_SLV_WR_STA_INTEN	SPI_SLV_RD_STA_INTEN	SPI_SLV_WR_BUF_INTEN	SPI_SLV_RD_BUF_INTEN	SPI_TRANS_DONE	SPI_SLV_WR_STA_DONE	SPI_SLV_RD_STA_DONE	SPI_SLV_WR_BUF_DONE	SPI_SLV_RD_BUF_DONE	
31	30	29	28	27	26 23	22 20	19 17	16 12	11 10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0 0 0 0	0 0 0	0 0 0	0 0 0 0 0	0 0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_SYNC_RESET This bit is used to enable software reset. When set, it resets the latched values of the SPI clock line, cs line and data lines. (R/W)

SPI_SLAVE_MODE This bit is used to set the mode of the SPI device. (R/W)

1: slave mode;

0: master mode.

SPI_SLV_WR_RD_BUF_EN Setting this bit enables the write and read buffer commands in slave mode. (R/W)

SPI_SLV_WR_RD_STA_EN Setting this bit enables the write and read status commands in slave mode. (R/W)

SPI_SLV_CMD_DEFINE This bit is used to enable custom slave mode commands. (R/W)

1: slave mode commands are defined in SPI_SLAVE3.

0: slave mode commands are fixed as: 0x1: write-status; 0x2: write-buffer, 0x3: read-buffer; and 0x4: read-status.

SPI_TRANS_CNT The counter for operations in both the master mode and the slave mode. (RO)

SPI_SLV_LAST_STATE In slave mode, this contains the state of the SPI state machine. (RO)

SPI_SLV_LAST_COMMAND In slave mode, this contains the value of the received command. (RO)

SPI_CS_I_MODE In the slave mode, this selects the mode to synchronize the input SPI cs signal and eliminate SPI cs jitter. (R/W)

0: configured through registers (SPI_CS_DELAY_NUM and SPI_CS_DELAY_MODE);

1: using double synchronization method and configured through registers (SPI_CS_DELAY_NUM and SPI_CS_DELAY_MODE);

2: using double synchronization method.

SPI_TRANS_INTEN The interrupt enable bit for the SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_SLV_WR_STA_INTEN The interrupt enable bit for the SPI_SLV_WR_STA_INT interrupt. (R/W)

SPI_SLV_RD_STA_INTEN The interrupt enable bit for the SPI_SLV_RD_STA_INT interrupt. (R/W)

SPI_SLV_WR_BUF_INTEN The interrupt enable bit for the SPI_SLV_WR_BUF_INT interrupt. (R/W)

SPI_SLV_RD_BUF_INTEN The interrupt enable bit for the SPI_SLV_RD_BUF_INT interrupt. (R/W)

SPI_TRANS_DONE The raw interrupt status bit for the SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_SLV_WR_STA_DONE The raw interrupt status bit for the SPI_SLV_WR_STA_INT interrupt. (R/W)

SPI_SLV_RD_STA_DONE The raw interrupt status bit for the SPI_SLV_RD_STA_INT interrupt. (R/W)

SPI_SLV_WR_BUF_DONE The raw interrupt status bit for the SPI_SLV_WR_BUF_INT interrupt. (R/W)

SPI_SLV_RD_BUF_DONE The raw interrupt status bit for the SPI_SLV_RD_BUF_INT interrupt. (R/W)

Register 7.16: SPI_SLAVE1_REG (0x3C)

SPI_SLV_STATUS_BITLEN	SPI_SLV_STATUS_FAST_EN	SPI_SLV_STATUS_READBACK	(reserved)	SPI_SLV_RD_ADDR_BITLEN	SPI_SLV_WR_ADDR_BITLEN	SPI_SLV_WRSTA_DUMMY_EN	SPI_SLV_RDSTA_DUMMY_EN	SPI_SLV_WRBUF_DUMMY_EN	SPI_SLV_RDBUF_DUMMY_EN	
31 27	26	25	24 16	15 10	9 4	3	2	1	0	
0 0 0 0 0	0	1	0 0 0 0 0 0 0 0 0	0x00	0x00	0	0	0	0	Reset

SPI_SLV_STATUS_BITLEN In slave mode, this sets the length of the status field. (R/W)

SPI_SLV_STATUS_FAST_EN In slave mode, this enables fast reads of the status. (R/W)

SPI_SLV_STATUS_READBACK In slave mode, this selects the active status register. (R/W)
1: reads register of SPI_SLV_WR_STATUS;
0: reads register of SPI_RD_STATUS.

SPI_SLV_RD_ADDR_BITLEN In slave mode, this contains the address length in bits for a read-buffer operation, minus one. (R/W)

SPI_SLV_WR_ADDR_BITLEN In slave mode, this contains the address length in bits for a write-buffer operation, minus one. (R/W)

SPI_SLV_WRSTA_DUMMY_EN In slave mode, this bit enables the dummy phase for write-status operations. (R/W)

SPI_SLV_RDSTA_DUMMY_EN In slave mode, this bit enables the dummy phase for read-status operations. (R/W)

SPI_SLV_WRBUF_DUMMY_EN In slave mode, this bit enables the dummy phase for write-buffer operations. (R/W)

SPI_SLV_RDBUF_DUMMY_EN In slave mode, this bit enables the dummy phase for read-buffer operations. (R/W)

Register 7.17: SPI_SLAVE2_REG (0x40)

SPI_SLV_WRBUF_DUMMY_CYCLELEN	SPI_SLV_RDBUF_DUMMY_CYCLELEN	SPI_SLV_WRSTA_DUMMY_CYCLELEN	SPI_SLV_RDSTA_DUMMY_CYCLELEN	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0x000	0x000	0x000	Reset

SPI_SLV_WRBUF_DUMMY_CYCLELEN In slave mode, this contains number of spi_clk cycles for the dummy phase for write-buffer operations, minus one. (R/W)

SPI_SLV_RDBUF_DUMMY_CYCLELEN In slave mode, this contains the number of spi_clk cycles for the dummy phase for read-buffer operations, minus one (R/W)

SPI_SLV_WRSTA_DUMMY_CYCLELEN In slave mode, this contains the number of spi_clk cycles for the dummy phase for write-status operations, minus one. (R/W)

SPI_SLV_RDSTA_DUMMY_CYCLELEN In slave mode, this contains the number of spi_clk cycles for the dummy phase for read-status operations, minus one. (R/W)

Register 7.18: SPI_SLAVE3_REG (0x44)

SPI_SLV_WRSTA_CMD_VALUE	SPI_SLV_RDSTA_CMD_VALUE	SPI_SLV_WRBUF_CMD_VALUE	SPI_SLV_RDBUF_CMD_VALUE	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Reset

SPI_SLV_WRSTA_CMD_VALUE In slave mode, this contains the value of the write-status command. (R/W)

SPI_SLV_RDSTA_CMD_VALUE In slave mode, this contains the value of the read-status command. (R/W)

SPI_SLV_WRBUF_CMD_VALUE In slave mode, this contains the value of the write-buffer command. (R/W)

SPI_SLV_RDBUF_CMD_VALUE In slave mode, this contains the value of the read-buffer command. (R/W)

Register 7.19: SPI_SLV_WRBUF_DLEN_REG (0x48)

(reserved)	SPI_SLV_WRBUF_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_SLV_WRBUF_DBITLEN This equals to the bit length of data written into the slave buffer, minus one. (R/W)

Register 7.20: SPI_SLV_RDBUF_DLEN_REG (0x4C)

(reserved)	SPI_SLV_RDBUF_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_SLV_RDBUF_DBITLEN This equals to the bit length of data read from the slave buffer, minus one. (R/W)

Register 7.21: SPI_SLV_RD_BIT_REG (0x64)

(reserved)	SPI_SLV_RDATA_BIT	
31 24	23 0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_SLV_RDATA_BIT This equals to the bit length of data the master reads from the slave, minus one. (R/W)

Register 7.22: SPI_W*n*_REG (*n*: 0-15) (0x80+4**n*)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_W*n*_REG Data buffer. (R/W)

Register 7.23: SPI_TX_CRC_REG (0xC0)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_TX_CRC_REG For SPI1, this contains the CRC32 value of 256 bits of data. (R/W)

Register 7.24: SPI_EXT2_REG (0xF8)

(reserved)	SPI_ST	
31 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	Reset

SPI_ST The current state of the SPI state machine: (RO)

0: idle state

1: preparation state

2: send command state

3: send data state

4: read data state

5: write data state

6: wait state

7: done state

Register 7.25: SPI_DMA_CONF_REG (0x100)

(reserved)	SPI_DMA_CONTINUE	SPI_DMA_TX_STOP	SPI_DMA_RX_STOP	(reserved)	SPI_OUT_DATA_BURST_EN	SPI_INDSCR_BURST_EN	SPI_OUTDSCR_BURST_EN	SPI_OUT_EOF_MODE	(reserved)	SPI_AHBM_RST	SPI_AHBM_FIFO_RST	SPI_OUT_RST	SPI_IN_RST	(reserved)	
31 17	16	15	14	13	12	11	10	9	8 6	5	4	3	2	3 2	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	1	0 0 0	0	0	0	0	0 0	Reset

SPI_DMA_CONTINUE This bit enables SPI DMA continuous data Tx/Rx mode. (R/W)

SPI_DMA_TX_STOP When in continuous Tx/Rx mode, setting this bit stops sending data. (R/W)

SPI_DMA_RX_STOP When in continuous Tx/Rx mode, setting this bit stops receiving data. (R/W)

SPI_OUT_DATA_BURST_EN SPI DMA reads data from memory in burst mode. (R/W)

SPI_INDSCR_BURST_EN SPI DMA reads descriptor in burst mode when writing data to the memory. (R/W)

SPI_OUTDSCR_BURST_EN SPI DMA reads descriptor in burst mode when reading data from the memory. (R/W)

SPI_OUT_EOF_MODE DMA out-EOF-flag generation mode. (R/W)
1: out-EOF-flag is generated when DMA has popped all data from the FIFO;
0: out-EOF-flag is generated when DMA has pushed all data to the FIFO.

SPI_AHBM_RST reset SPI DMA AHB master. (R/W)

SPI_AHBM_FIFO_RST This bit is used to reset SPI DMA AHB master FIFO pointer. (R/W)

SPI_OUT_RST The bit is used to reset DMA out-FSM and out-data FIFO pointer. (R/W)

SPI_IN_RST The bit is used to reset DMA in-DSM and in-data FIFO pointer. (R/W)

Register 7.26: SPI_DMA_OUT_LINK_REG (0x104)

(reserved)	SPI_OUTLINK_RESTART	SPI_OUTLINK_START	SPI_OUTLINK_STOP	(reserved)	SPI_OUTLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

SPI_OUTLINK_RESTART Set the bit to add new outlink descriptors. (R/W)

SPI_OUTLINK_START Set the bit to start to use outlink descriptor. (R/W)

SPI_OUTLINK_STOP Set the bit to stop to use outlink descriptor. (R/W)

SPI_OUTLINK_ADDR The address of the first outlink descriptor. (R/W)

Register 7.27: SPI_DMA_IN_LINK_REG (0x108)

(reserved)	SPI_INLINK_RESTART	SPI_INLINK_START	SPI_INLINK_STOP	(reserved)	SPI_INLINK_AUTO_RET	SPI_INLINK_ADDR	
31	30	29	28	27 21	20	19 0	
0	0	0	0	0 0 0 0 0 0 0	0	0x000000	Reset

SPI_INLINK_RESTART Set the bit to add new inlink descriptors. (R/W)

SPI_INLINK_START Set the bit to start to use inlink descriptor. (R/W)

SPI_INLINK_STOP Set the bit to stop to use inlink descriptor. (R/W)

SPI_INLINK_AUTO_RET when the bit is set, inlink descriptor jumps to the next descriptor when a packet is invalid. (R/W)

SPI_INLINK_ADDR The address of the first inlink descriptor. (R/W)

Register 7.28: SPI_DMA_STATUS_REG (0x10C)

(reserved)	SPI_DMA_TX_EN	SPI_DMA_RX_EN	
31 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

SPI_DMA_TX_EN SPI DMA write-data status bit. (RO)

SPI_DMA_RX_EN SPI DMA read-data status bit. (RO)

Register 7.29: SPI_DMA_INT_ENA_REG (0x110)

(reserved)	SPI_OUT_TOTAL_EOF_INT_ENA	SPI_OUT_EOF_INT_ENA	SPI_OUT_DONE_INT_ENA	SPI_IN_SUC_EOF_INT_ENA	SPI_IN_ERR_EOF_INT_ENA	SPI_IN_DONE_INT_ENA	SPI_INLINK_DSCR_ERROR_INT_ENA	SPI_OUTLINK_DSCR_ERROR_INT_ENA	SPI_INLINK_DSCR_EMPTY_INT_ENA	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_ENA The interrupt enable bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (R/W)

SPI_OUT_EOF_INT_ENA The interrupt enable bit for the SPI_OUT_EOF_INT interrupt. (R/W)

SPI_OUT_DONE_INT_ENA The interrupt enable bit for the SPI_OUT_DONE_INT interrupt. (R/W)

SPI_IN_SUC_EOF_INT_ENA The interrupt enable bit for the SPI_IN_SUC_EOF_INT interrupt. (R/W)

SPI_IN_ERR_EOF_INT_ENA The interrupt enable bit for the SPI_IN_ERR_EOF_INT interrupt. (R/W)

SPI_IN_DONE_INT_ENA The interrupt enable bit for the SPI_IN_DONE_INT interrupt. (R/W)

SPI_INLINK_DSCR_ERROR_INT_ENA The interrupt enable bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (R/W)

SPI_OUTLINK_DSCR_ERROR_INT_ENA The interrupt enable bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (R/W)

SPI_INLINK_DSCR_EMPTY_INT_ENA The interrupt enable bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (R/W)

Register 7.30: SPI_DMA_INT_RAW_REG (0x114)

(reserved)	SPI_OUT_TOTAL_EOF_INT_RAW	SPI_OUT_EOF_INT_RAW	SPI_OUT_DONE_INT_RAW	SPI_IN_SUC_EOF_INT_RAW	SPI_IN_ERR_EOF_INT_RAW	SPI_IN_DONE_INT_RAW	SPI_INLINK_DSCR_ERROR_INT_RAW	SPI_OUTLINK_DSCR_ERROR_INT_RAW	SPI_INLINK_DSCR_EMPTY_INT_RAW	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_RAW The raw interrupt status bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (RO)

SPI_OUT_EOF_INT_RAW The raw interrupt status bit for the SPI_OUT_EOF_INT interrupt. (RO)

SPI_OUT_DONE_INT_RAW The raw interrupt status bit for the SPI_OUT_DONE_INT interrupt. (RO)

SPI_IN_SUC_EOF_INT_RAW The raw interrupt status bit for the SPI_IN_SUC_EOF_INT interrupt. (RO)

SPI_IN_ERR_EOF_INT_RAW The raw interrupt status bit for the SPI_IN_ERR_EOF_INT interrupt. (RO)

SPI_IN_DONE_INT_RAW The raw interrupt status bit for the SPI_IN_DONE_INT interrupt. (RO)

SPI_INLINK_DSCR_ERROR_INT_RAW The raw interrupt status bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (RO)

SPI_OUTLINK_DSCR_ERROR_INT_RAW The raw interrupt status bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (RO)

SPI_INLINK_DSCR_EMPTY_INT_RAW The raw interrupt status bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (RO)

Register 7.31: SPI_DMA_INT_ST_REG (0x118)

(reserved)	SPI_OUT_TOTAL_EOF_INT_ST	SPI_OUT_EOF_INT_ST	SPI_OUT_DONE_INT_ST	SPI_IN_SUC_EOF_INT_ST	SPI_IN_ERR_EOF_INT_ST	SPI_IN_DONE_INT_ST	SPI_INLINK_DSCR_ERROR_INT_ST	SPI_OUTLINK_DSCR_ERROR_INT_ST	SPI_INLINK_DSCR_EMPTY_INT_ST	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_ST The masked interrupt status bit for the SPI_OUT_TOTAL_EOF_INT interrupt. (RO)

SPI_OUT_EOF_INT_ST The masked interrupt status bit for the

SPI_OUT_EOF_INT interrupt. (RO)

SPI_OUT_DONE_INT_ST The masked interrupt status bit for the SPI_OUT_DONE_INT interrupt. (RO)

SPI_IN_SUC_EOF_INT_ST The masked interrupt status bit for the SPI_IN_SUC_EOF_INT interrupt. (RO)

SPI_IN_ERR_EOF_INT_ST The masked interrupt status bit for the SPI_IN_ERR_EOF_INT interrupt. (RO)

SPI_IN_DONE_INT_ST The masked interrupt status bit for the SPI_IN_DONE_INT interrupt. (RO)

SPI_INLINK_DSCR_ERROR_INT_ST The masked interrupt status bit for the SPI_INLINK_DSCR_ERROR_INT interrupt. (RO)

SPI_OUTLINK_DSCR_ERROR_INT_ST The masked interrupt status bit for the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (RO)

SPI_INLINK_DSCR_EMPTY_INT_ST The masked interrupt status bit for the SPI_INLINK_DSCR_EMPTY_INT interrupt. (RO)

Register 7.32: SPI_DMA_INT_CLR_REG (0x11C)

(reserved)	SPI_OUT_TOTAL_EOF_INT_CLR	SPI_OUT_EOF_INT_CLR	SPI_OUT_DONE_INT_CLR	SPI_IN_SUC_EOF_INT_CLR	SPI_IN_ERR_EOF_INT_CLR	SPI_IN_DONE_INT_CLR	SPI_INLINK_DSCR_ERROR_INT_CLR	SPI_OUTLINK_DSCR_ERROR_INT_CLR	SPI_INLINK_DSCR_EMPTY_INT_CLR	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_CLR Set this bit to clear the SPI_OUT_TOTAL_EOF_INT interrupt. (R/W)

SPI_OUT_EOF_INT_CLR Set this bit to clear the SPI_OUT_EOF_INT interrupt. (R/W)

SPI_OUT_DONE_INT_CLR Set this bit to clear the SPI_OUT_DONE_INT interrupt. (R/W)

SPI_IN_SUC_EOF_INT_CLR Set this bit to clear the SPI_IN_SUC_EOF_INT interrupt. (R/W)

SPI_IN_ERR_EOF_INT_CLR Set this bit to clear the SPI_IN_ERR_EOF_INT interrupt. (R/W)

SPI_IN_DONE_INT_CLR Set this bit to clear the SPI_IN_DONE_INT interrupt. (R/W)

SPI_INLINK_DSCR_ERROR_INT_CLR Set this bit to clear the SPI_INLINK_DSCR_ERROR_INT interrupt. (R/W)

SPI_OUTLINK_DSCR_ERROR_INT_CLR Set this bit to clear the SPI_OUTLINK_DSCR_ERROR_INT interrupt. (R/W)

SPI_INLINK_DSCR_EMPTY_INT_CLR Set this bit to clear the SPI_INLINK_DSCR_EMPTY_INT interrupt. (R/W)

Register 7.33: SPI_IN_ERR_EOF_DES_ADDR_REG (0x120)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_IN_ERR_EOF_DES_ADDR_REG The inlink descriptor address when SPI DMA encountered an error in receiving data. (RO)

Register 7.34: SPI_IN_SUC_EOF_DES_ADDR_REG (0x124)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_IN_SUC_EOF_DES_ADDR_REG The last inlink descriptor address when SPI DMA encountered EOF. (RO)

Register 7.35: SPI_INLINK_DSCR_REG (0x128)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_INLINK_DSCR_REG The address of the current inlink descriptor. (RO)

Register 7.36: SPI_INLINK_DSCR_BF0_REG (0x12C)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_INLINK_DSCR_BF0_REG The address of the next inlink descriptor. (RO)

Register 7.37: SPI_INLINK_DSCR_BF1_REG (0x130)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_INLINK_DSCR_BF1_REG The address of the next inlink data buffer. (RO)

Register 7.38: SPI_OUT_EOF_BFR_DES_ADDR_REG (0x134)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUT_EOF_BFR_DES_ADDR_REG The buffer address corresponding to the outlink descriptor that produces EOF. (RO)

Register 7.39: SPI_OUT_EOF_DES_ADDR_REG (0x138)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUT_EOF_DES_ADDR_REG The last outlink descriptor address when SPI DMA encountered EOF. (RO)

Register 7.40: SPI_OUTLINK_DSCR_REG (0x13C)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUTLINK_DSCR_REG The address of the current outlink descriptor. (RO)

Register 7.41: SPI_OUTLINK_DSCR_BF0_REG (0x140)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUTLINK_DSCR_BF0_REG The address of the next outlink descriptor. (RO)

Register 7.42: SPI_OUTLINK_DSCR_BF1_REG (0x144)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUTLINK_DSCR_BF1_REG The address of the next outlink data buffer. (RO)

Register 7.43: SPI_DMA_RSTATUS_REG (0x148)

TX_FIFO_EMPTY	TX_FIFO_FULL	(reserved)	TX_DES_ADDRESS	
31	30	29 20	19 0	
0	0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

TX_FIFO_EMPTY The SPI DMA Tx FIFO is empty. (RO)

TX_FIFO_FULL The SPI DMA Tx FIFO is full. (RO)

TX_DES_ADDRESS The LSB of the SPI DMA outlink descriptor address. (RO)

Register 7.44: SPI_DMA_TSTATUS_REG (0x14C)

RX_FIFO_EMPTY	RX_FIFO_FULL	(reserved)	RX_DES_ADDRESS	
31	30	29 20	19 0	
0	0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RX_FIFO_EMPTY The SPI DMA Rx FIFO is empty. (RO)

RX_FIFO_FULL The SPI DMA Rx FIFO is full. (RO)

RX_DES_ADDRESS The LSB of the SPI DMA inlink descriptor address. (RO)

8. SDIO Slave

8.1 Overview

The ESP32 features hardware support for the industry-standard Secure Digital (SD) device interface that conforms to the SD Input/Output (SDIO) Specification Version 2.0. This allows a host controller to access the ESP32 via an SDIO bus protocol, enabling high-speed data transfer.

The SDIO interface may be used to read ESP32 SDIO registers directly and access shared memory via Direct Memory Access (DMA), thus reducing processing overhead while maintaining high performance.

8.2 Features

- Meets SDIO V2.0 specification
- Supports SDIO SPI, 1-bit, and 4-bit transfer modes
- Full host clock range of 0 ~ 50 MHz
- Configurable sample and drive clock edge
- Integrated, SDIO-accessible registers for information interaction
- Supports SDIO interrupt mechanism
- Automatic data padding
- Block size of up to 512 bytes
- Interrupt vector between Host and Slave for bidirectional interrupt
- Supports DMA for data transfer

8.3 Functional Description

8.3.1 SDIO Slave Block Diagram

The functional block diagram of the SDIO slave module is shown in Figure 21.

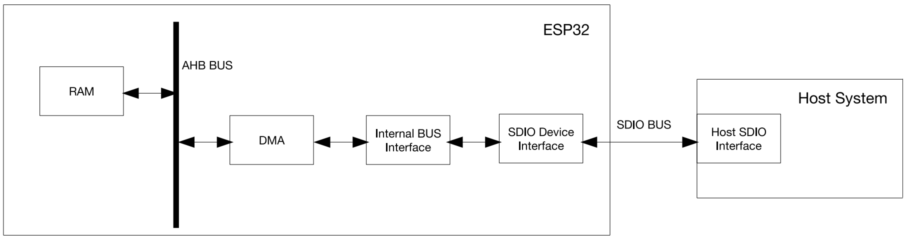


Figure 21: SDIO Slave Block Diagram

The Host System represents any SDIO specification V2.0-compatible host device. The Host System interacts with the ESP32 (configured as the SDIO slave) via the standard SDIO bus implementation.

The SDIO Device Interface block enables effective communication with the external Host by directly providing SDIO interface registers and enabling DMA operation for high-speed data transfer over the Advanced High-performance Bus (AHB) without engaging the CPU.

8.3.2 Sending and Receiving Data on SDIO Bus

Data is transmitted between Host and Slave through the SDIO bus I/O Function1. After the Host enables the I/O Function1 in the Slave, according to the SDIO protocol, data transmission will begin.

ESP32 segregates data into packets sent to/from the Host. To achieve high bus utilization and data transfer rates, we recommend the single block transmission mode. For detailed information on this mode, please refer to the SDIO V2.0 protocol specification. When Host and Slave exchange data as blocks on the SDIO bus, the Slave automatically pads data-when sending data out-and automatically strips padding data from the incoming data block.

Whether the Slave pads or discards the data depends on the data address on the SDIO bus. When the data address is equal to, or greater than, 0x1F800, the Slave will start padding or discarding data. Therefore, the starting data address should be 0x1F800 - Packet_length, where Packet_length is measured in bytes. Data flow on the SDIO bus is shown in Figure 22.

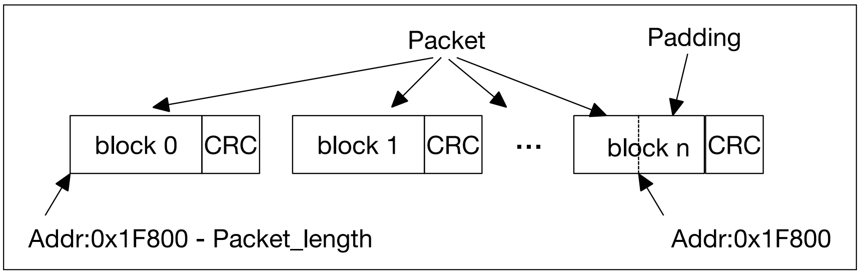


Figure 22: SDIO Bus Packet Transmission

The standard IO_RW_EXTENDED (CMD53) command is used to initiate a packet transfer of an arbitrary length. The content of the CMD53 command used in data transmission is as illustrated in Figure 23 below. For detailed information on CMD53, please refer to the SDIO protocol specifications.

S	D	Command Index 110101b	R/W flag	Function Number 001b	Block Mode 1b	OP Code 1b	Register Address 0X1F800-Packet_length	Byte/Block Count Roundup (Packet_length/Block_size)	CRC7	E
1	1	6	1	3	1	1	17	9	7	1

Figure 23: CMD53 Content

8.3.3 Register Access

For effective interaction between Host and Slave, the Host can access certain registers in the Slave via the SDIO bus I/O Function1. These registers are in continuous address fields from SLC0HOST_TOKEN_RDATA to SLCHOST_INF_ST. The Host device can access these registers by simply setting the register addresses of CMD52 or CMD53 to the low 10 bits of the corresponding register address. The Host can access several consecutive registers at one go with CMD53, thus achieving a higher effective transfer rate.

There are 54 bytes of field between SLCHOST_CONF_W0_REG and SLCHOST_CONF_W15_REG. Host and Slave can access and change these fields, thus facilitating the information interaction between Host and Slave.

8.3.4 DMA

The SDIO Slave module uses dedicated DMA to access data residing in the RAM. As shown in Figure 21, the RAM is accessed over the AHB. DMA accesses RAM through a linked-list descriptor. Every linked list is composed of three words, as shown in Figure 24.

1	1	6	12	12
Owner	Eof	Reserved	Length	Size
Buffer Address Pointer				
Next Descriptor Address				

Figure 24: SDIO Slave DMA Linked List Structure

- Owner: The allowed operator of the buffer that corresponds to the current linked list. 0: CPU is the allowed operator; 1: DMA is the allowed operator.
- Eof: End-of-file marker, indicating that this linked-list element is the last element of the data packet.
- Length: The number of valid bytes in the buffer, i.e., the number of bytes that should be accessed from the buffer for reading/writing.
- Size: The maximum number of available buffers.
- Buffer Address Pointer: The address of the data buffer as seen by the CPU (according to the RAM address space).
- Next Descriptor Address: The address of the next linked-list element in the CPU RAM address space. If the current linked list is the last one, the Eof bit should be 1, and the last descriptor address should be 0.

The Slave's linked-list chain is shown in Figure 25:

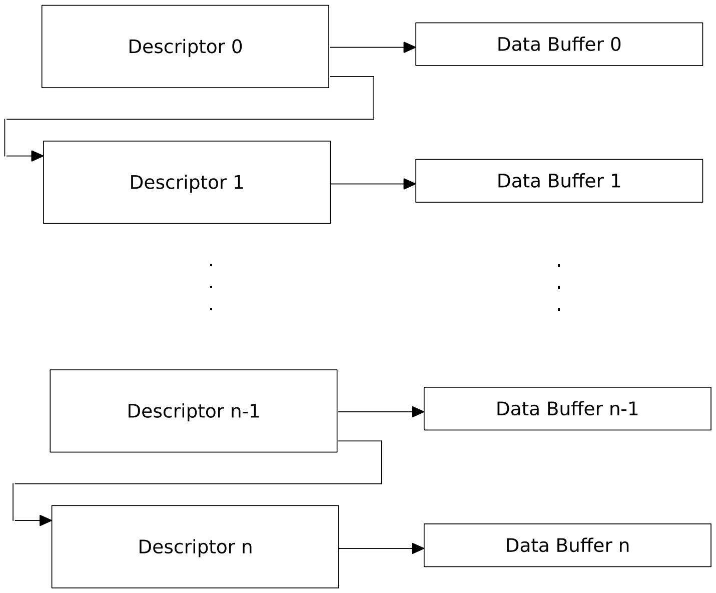


Figure 25: SDIO Slave Linked List

8.3.5 Packet-Sending/-Receiving Procedure

The SDIO Host and Slave devices need to follow specific data transfer procedures to successfully exchange data over the SDIO interface.

8.3.5.1 Sending Packets to SDIO Host

The transmission of packets from Slave to Host is initiated by the Slave. The Host will be notified with an interrupt (for detailed information on interrupts, please refer to SDIO protocol). After the Host reads the relevant information from the Slave, it will initiate an SDIO bus transaction accordingly. The whole procedure is illustrated in Figure 26.

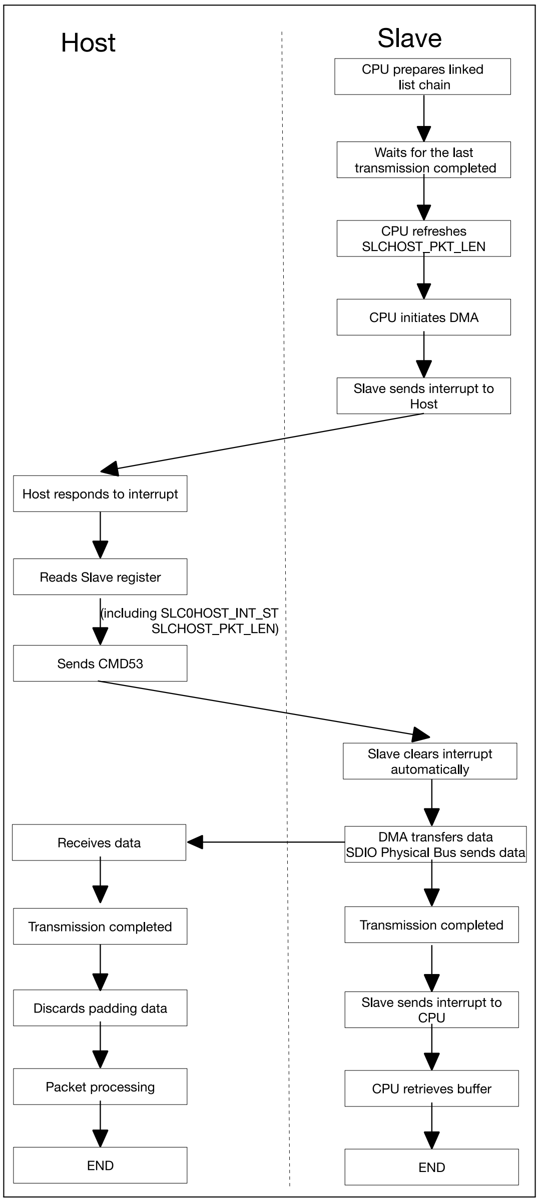

Host
Slave
CPU prepares linked list chain
Waits for the last transmission completed
CPU refreshes SLCHOST_PKT_LEN
CPU initiates DMA
Slave sends interrupt to Host
Host responds to interrupt
Reads Slave register
(including SLC0HOST_INT_ST SLCHOST_PKT_LEN)
Sends CMD53
Slave clears interrupt automatically
Receives data
DMA transfers data SDIO Physical Bus sends data
Transmission completed
Transmission completed
Discards padding data
Slave sends interrupt to CPU
Packet processing
CPU retrieves buffer
END
END

Figure 26: Packet Sending Procedure (Initiated by Slave)

When the Host is interrupted, it reads relevant information from the Slave by visiting registers SLC0HOST_INT and SLCHOST_PKT_LEN.

- SLC0HOST_INT: Interrupt status register. If the value of SLC0_RX_NEW_PACKET_INT_ST is 1, this indicates that the Slave has a packet to send.
- SLCHOST_PKT_LEN: Packet length accumulator register. The current value minus the value of last time equals the packet length sent this time.

In order to start DMA, the CPU needs to write the low 20 bits of the address of the first linked-list element to the SLC0_RXLINK_ADDR bit of SLC0RX_LINK, then set the SLC0_RXLINK_START bit of SLC0RX_LINK. The DMA will automatically complete the data transfer. Upon completion of the operation, DMA will interrupt the CPU so that the buffer space can be freed or reused.

8.3.5.2 Receiving Packets from SDIO Host

Transmission of packets from Host to Slave is initiated by the Host. The Slave receives data via DMA and stores it in RAM. After transmission is completed, the CPU will be interrupted to process the data. The whole procedure is demonstrated in Figure 27.

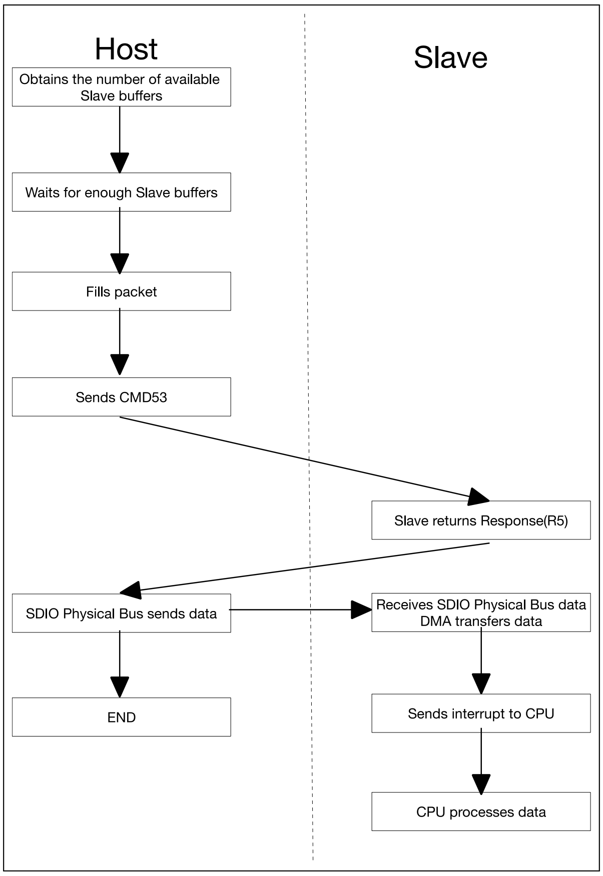

Host
Slave
Obtains the number of available Slave buffers
Waits for enough Slave buffers
Fills packet
Sends CMD53
Slave returns Response(R5)
SDIO Physical Bus sends data
Receives SDIO Physical Bus data DMA transfers data
END
Sends interrupt to CPU
CPU processes data

Figure 27: Packet Receiving Procedure (Initiated by Host)

The Host obtains the number of available receiving buffers from the Slave by accessing register SLC0HOST_TOKEN_RDATA. The Slave CPU should update this value after the receiving DMA linked list is prepared.

HOSTREG_SLC0_TOKEN1 in SLC0HOST_TOKEN_RDATA stores the accumulated number of available buffers.

The Host can figure out the available buffer space, using HOSTREG_SLC0_TOKEN1 minus the number of buffers already used.

If the buffers are not enough, the Host needs to constantly poll the register until there are enough buffers available.

To ensure sufficient receiving buffers, the Slave CPU must constantly load buffers on the receiving linked list. The process is shown in Figure 28.

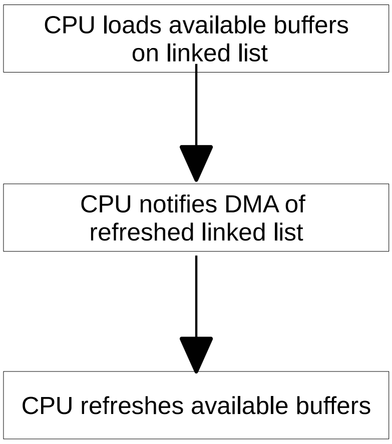

CPU loads available buffers on linked list
CPU notifies DMA of refreshed linked list
CPU refreshes available buffers

Figure 28: Loading Receiving Buffer

The CPU first needs to append new buffer segments at the end of the linked list that is being used by DMA and is available for receiving data.

The CPU then needs to notify the DMA that the linked list has been modified. This can be done by setting bit SLC0_TXLINK_RESTART of the SLC0TX_LINK register. Please note that when the CPU initiates DMA to receive packets for the first time, SLC0_TXLINK_RESTART should be set to 1.

Lastly, the CPU refreshes any available buffer information by writing to the SLC0TOKEN1 register.

8.3.6 SDIO Bus Timing

The SDIO bus operates at a very high speed and the PCB trace length usually affects signal integrity by introducing latency. To ensure that the timing characteristics conform to the desired bus timing, the SDIO Slave module supports configuration of input sampling clock edge and output driving clock edge.

When the incoming data changes near the rising edge of the clock, the Slave will perform sampling on the falling edge of the clock, or vice versa, as Figure 29 shows.

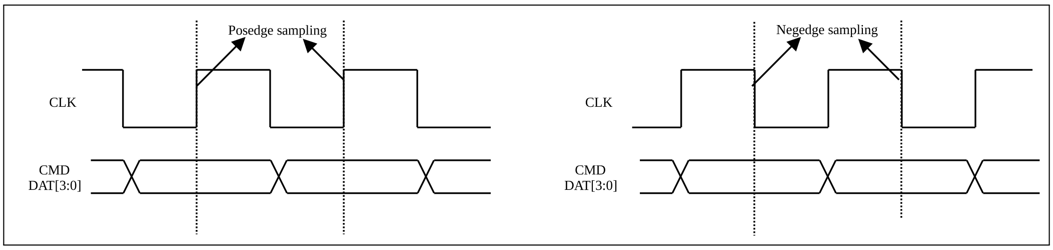


Figure 29: Sampling Timing Diagram

Sampling edges are configured via the FRC_POS_SAMP and FRC_NEG_SAMP bitfields in the SLCHOST_CONF register. Each field is five bits wide, with bits corresponding to the CMD line and four DATA lines (0-3). Setting a bit in FRC_POS_SAMP causes the corresponding line to be sampled for input at the rising clock edge, whereas setting a bit in FRC_NEG_SAMP causes the corresponding line to be sampled for input at the falling clock edge.

The Slave can also select the edge at which data output lines are driven to accommodate for any latency caused by the physical signal path, as shown in Figure 30.

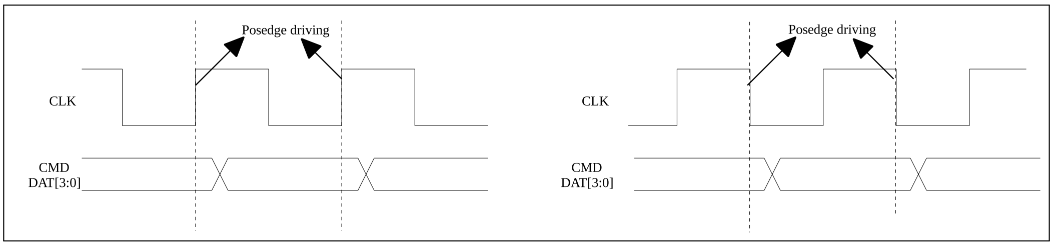


Figure 30: Output Timing Diagram

Driving edges are configured via the FRC_SDIO20 and FRC_SDIO11 bitfields in the SLCHOST_CONF register. Each field is five bits wide, with bits corresponding to the CMD line and four DATA lines (0-3). Setting a bit in FRC_SDIO20 causes the corresponding line to output at the rising clock edge, whereas setting a bit in FRC_SDIO11 causes the corresponding line to output at the falling clock edge.

8.3.7 Interrupt

Host and Slave can interrupt each other via the interrupt vector. Both Host and Slave have eight interrupt vectors. The interrupt is enabled by configuring the interrupt vector register (setting the enable bit to 1). The interrupt vector registers can clear themselves automatically, which means one interrupt at a time and no other configuration is required.

8.3.7.1 Host Interrupt

- *SLC0HOST_SLC0_RX_NEW_PACKET_INT* Slave has a packet to send.
- *SLC0HOST_SLC0_TX_OVF_INT* Slave receiving buffer overflow interrupt.
- *SLC0HOST_SLC0_RX_UDF_INT* Slave sending buffer underflow interrupt.
- *SLC0HOST_SLC0_TOHOST_BITn_INT* (*n*: 0 ~ 7) Slave interrupts Host.

8.3.7.2 Slave Interrupt

- *SLC0INT_SLC0_RX_DSCR_ERR_INT* Slave sending descriptor error.
- *SLC0INT_SLC0_TX_DSCR_ERR_INT* Slave receiving descriptor error.
- *SLC0INT_SLC0_RX_EOF_INT* Slave sending operation is finished.
- *SLC0INT_SLC0_RX_DONE_INT* A single buffer is sent by Slave.
- *SLC0INT_SLC0_TX_SUC_EOF_INT* Slave receiving operation is finished.
- *SLC0INT_SLC0_TX_DONE_INT* A single buffer is finished during receiving operation.
- *SLC0INT_SLC0_TX_OVF_INT* Slave receiving buffer overflow interrupt.
- *SLC0INT_SLC0_RX_UDF_INT* Slave sending buffer underflow interrupt.
- *SLC0INT_SLC0_TX_START_INT* Slave receiving interrupt initialization.
- *SLC0INT_SLC0_RX_START_INT* Slave sending interrupt initialization.

- *SLC0INT_SLC_FRHOST_BITn_INT* (*n*: 0 ~ 7) Host interrupts Slave.

8.4 Register Summary

Name	Description	Address	Access
SDIO DMA (SLC) configuration registers			
SLCCONF0_REG	SLCCONF0_SLC configuration	0x3FF58000	R/W
SLC0INT_RAW_REG	Raw interrupt status	0x3FF58004	RO
SLC0INT_ST_REG	Interrupt status	0x3FF58008	RO
SLC0INT_ENA_REG	Interrupt enable	0x3FF5800C	R/W
SLC0INT_CLR_REG	Interrupt clear	0x3FF58010	WO
SLC0RX_LINK_REG	Transmitting linked list configuration	0x3FF5803C	R/W
SLC0TX_LINK_REG	Receiving linked list configuration	0x3FF58040	R/W
SLCINTVEC_TOHOST_REG	Interrupt sector for Slave to interrupt Host	0x3FF5804C	WO
SLC0TOKEN1_REG	Number of receiving buffer	0x3FF58054	WO
SLCCONF1_REG	Control register	0x3FF58060	R/W
SLC_RX_DSCR_CONF_REG	DMA transmission configuration	0x3FF58098	R/W
SLC0_LEN_CONF_REG	Length control of the transmitting packets	0x3FF580E4	R/W
SLC0_LENGTH_REG	Length of the transmitting packets	0x3FF580E8	R/W

Name	Description	Address	Access
SDIO SLC Host registers			
SLC0HOST_INT_RAW_REG	Raw interrupt	0x3FF55000	RO
SLC0HOST_TOKEN_RDATA	The accumulated number of Slave's receiving buffers	0x3FF55044	RO
SLC0HOST_INT_ST_REG	Masked interrupt status	0x3FF55058	RO
SLCHOST_PKT_LEN_REG	Length of the transmitting packets	0x3FF55060	RO
SLCHOST_CONF_W0_REG	Host and Slave communication register0	0x3FF5506C	R/W
SLCHOST_CONF_W1_REG	Host and Slave communication register1	0x3FF55070	R/W
SLCHOST_CONF_W2_REG	Host and Slave communication register2	0x3FF55074	R/W
SLCHOST_CONF_W3_REG	Host and Slave communication register3	0x3FF55078	R/W
SLCHOST_CONF_W4_REG	Host and Slave communication register4	0x3FF5507C	R/W
SLCHOST_CONF_W6_REG	Host and Slave communication register6	0x3FF55088	R/W
SLCHOST_CONF_W7_REG	Interrupt vector for Host to interrupt Slave	0x3FF5508C	WO
SLCHOST_CONF_W8_REG	Host and Slave communication register8	0x3FF5509C	R/W
SLCHOST_CONF_W9_REG	Host and Slave communication register9	0x3FF550A0	R/W
SLCHOST_CONF_W10_REG	Host and Slave communication register10	0x3FF550A4	R/W
SLCHOST_CONF_W11_REG	Host and Slave communication register11	0x3FF550A8	R/W
SLCHOST_CONF_W12_REG	Host and Slave communication register12	0x3FF550AC	R/W
SLCHOST_CONF_W13_REG	Host and Slave communication register13	0x3FF550B0	R/W
SLCHOST_CONF_W14_REG	Host and Slave communication register14	0x3FF550B4	R/W
SLCHOST_CONF_W15_REG	Host and Slave communication register15	0x3FF550B8	R/W
SLC0HOST_INT_CLR_REG	Interrupt clear	0x3FF550D4	WO
SLC0HOST_FUNC1_INT_ENA_REG	Interrupt enable	0x3FF550DC	R/W

SLCHOST_CONF_REG	Edge configuration	0x3FF551F0	R/W

Name	Description	Address	Access
SDIO HINF registers			
HINF_CFG_DATA1_REG	SDIO specification configuration	0x3FF4B004	R/W

8.5 SLC Registers

The first block of SDIO control registers starts at 0x3FF5_8000.

Register 8.1: SLCCONF0_REG (0x0)

(reserved)	SLCCONF0_SLC0_TOKEN_AUTO_CLR	(reserved)	SLCCONF0_SLC0_RX_AUTO_WRBACK	SLCCONF0_SLC0_RX_LOOP_TEST	SLCCONF0_SLC0_TX_LOOP_TEST	(reserved)	SLCCONF0_SLC0_RX_RST	SLCCONF0_SLC0_TX_RST	
31 15	14	13 7	6	5	4	3 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0 0 0 0 0	0	1	1	0 0	0	0	Reset

SLCCONF0_SLC0_TOKEN_AUTO_CLR Please initialize to 0. Do not modify it. (R/W)

SLCCONF0_SLC0_RX_AUTO_WRBACK Allows changing the owner bit of the transmitting buffer's linked list when transmitting data. (R/W)

SLCCONF0_SLC0_RX_LOOP_TEST Loop around when the slave buffer finishes sending packets. When set to 1, hardware will not change the owner bit in the linked list. (R/W)

SLCCONF0_SLC0_TX_LOOP_TEST Loop around when the slave buffer finishes receiving packets. When set to 1, hardware will not change the owner bit in the linked list. (R/W)

SLCCONF0_SLC0_RX_RST Set this bit to reset the transmitting FSM. (R/W)

SLCCONF0_SLC0_TX_RST Set this bit to reset the receiving FSM. (R/W)

Register 8.2: SLC0INT_RAW_REG (0x4)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_RAW	SLC0INT_SLC0_TX_DSCR_ERR_INT_RAW	(reserved)	SLC0INT_SLC0_RX_EOF_INT_RAW	SLC0INT_SLC0_RX_DONE_INT_RAW	SLC0INT_SLC0_TX_SUC_EOF_INT_RAW	SLC0INT_SLC0_TX_DONE_INT_RAW	(reserved)	SLC0INT_SLC0_TX_OVF_INT_RAW	SLC0INT_SLC0_RX_UDF_INT_RAW	SLC0INT_SLC0_TX_START_INT_RAW	SLC0INT_SLC0_RX_START_INT_RAW	SLC0INT_SLC_FRHOST_BIT7_INT_RAW	SLC0INT_SLC_FRHOST_BIT6_INT_RAW	SLC0INT_SLC_FRHOST_BIT5_INT_RAW	SLC0INT_SLC_FRHOST_BIT4_INT_RAW	SLC0INT_SLC_FRHOST_BIT3_INT_RAW	SLC0INT_SLC_FRHOST_BIT2_INT_RAW	SLC0INT_SLC_FRHOST_BIT1_INT_RAW	SLC0INT_SLC_FRHOST_BIT0_INT_RAW	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SLC0INT_SLC0_RX_DSCR_ERR_INT_RAW The raw interrupt bit for Slave sending descriptor error (RO)

SLC0INT_SLC0_TX_DSCR_ERR_INT_RAW The raw interrupt bit for Slave receiving descriptor error. (RO)

SLC0INT_SLC0_RX_EOF_INT_RAW The interrupt mark bit when Slave sending operation is finished. (RO)

SLC0INT_SLC0_RX_DONE_INT_RAW The raw interrupt bit to mark single buffer as sent by Slave. (RO)

SLC0INT_SLC0_TX_SUC_EOF_INT_RAW The raw interrupt bit to mark Slave receiving operation as finished. (RO)

SLC0INT_SLC0_TX_DONE_INT_RAW The raw interrupt bit to mark a single buffer as finished during Slave receiving operation. (RO)

SLC0INT_SLC0_TX_OVF_INT_RAW The raw interrupt bit to mark Slave receiving buffer overflow. (RO)

SLC0INT_SLC0_RX_UDF_INT_RAW The raw interrupt bit for Slave sending buffer underflow. (RO)

SLC0INT_SLC0_TX_START_INT_RAW The raw interrupt bit for registering Slave receiving initialization interrupt. (RO)

SLC0INT_SLC0_RX_START_INT_RAW The raw interrupt bit to mark Slave sending initialization interrupt. (RO)

SLC0INT_SLC_FRHOST_BIT7_INT_RAW The interrupt mark bit 7 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT6_INT_RAW The interrupt mark bit 6 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT5_INT_RAW The interrupt mark bit 5 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT4_INT_RAW The interrupt mark bit 4 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT3_INT_RAW The interrupt mark bit 3 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT2_INT_RAW The interrupt mark bit 2 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT1_INT_RAW The interrupt mark bit 1 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT0_INT_RAW The interrupt mark bit 0 for Host to interrupt Slave. (RO)

Register 8.3: SLC0INT_ST_REG (0x8)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_ST	SLC0INT_SLC0_TX_DSCR_ERR_INT_ST	(reserved)	SLC0INT_SLC0_RX_EOF_INT_ST	SLC0INT_SLC0_RX_DONE_INT_ST	SLC0INT_SLC0_TX_SUC_EOF_INT_ST	SLC0INT_SLC0_TX_DONE_INT_ST	(reserved)	SLC0INT_SLC0_TX_OVF_INT_ST	SLC0INT_SLC0_RX_UDF_INT_ST	SLC0INT_SLC0_TX_START_INT_ST	SLC0INT_SLC0_RX_START_INT_ST	SLC0INT_SLC_FRHOST_BIT7_INT_ST	SLC0INT_SLC_FRHOST_BIT6_INT_ST	SLC0INT_SLC_FRHOST_BIT5_INT_ST	SLC0INT_SLC_FRHOST_BIT4_INT_ST	SLC0INT_SLC_FRHOST_BIT3_INT_ST	SLC0INT_SLC_FRHOST_BIT2_INT_ST	SLC0INT_SLC_FRHOST_BIT1_INT_ST	SLC0INT_SLC_FRHOST_BIT0_INT_ST	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SLC0INT_SLC0_RX_DSCR_ERR_INT_ST The interrupt status bit for Slave sending descriptor error. (RO)

SLC0INT_SLC0_TX_DSCR_ERR_INT_ST The interrupt status bit for Slave receiving descriptor error. (RO)

SLC0INT_SLC0_RX_EOF_INT_ST The interrupt status bit for finished Slave sending operation. (RO)

SLC0INT_SLC0_RX_DONE_INT_ST The interrupt status bit for finished Slave sending operation. (RO)

SLC0INT_SLC0_TX_SUC_EOF_INT_ST The interrupt status bit for marking Slave receiving operation as finished. (RO)

SLC0INT_SLC0_TX_DONE_INT_ST The interrupt status bit for marking a single buffer as finished during the receiving operation. (RO)

SLC0INT_SLC0_TX_OVF_INT_ST The interrupt status bit for Slave receiving overflow interrupt. (RO)

SLC0INT_SLC0_RX_UDF_INT_ST The interrupt status bit for Slave sending buffer underflow. (RO)

SLC0INT_SLC0_TX_START_INT_ST The interrupt status bit for Slave receiving interrupt initialization. (RO)

SLC0INT_SLC0_RX_START_INT_ST The interrupt status bit for Slave sending interrupt initialization. (RO)

SLC0INT_SLC_FRHOST_BIT7_INT_ST The interrupt status bit 7 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT6_INT_ST The interrupt status bit 6 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT5_INT_ST The interrupt status bit 5 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT4_INT_ST The interrupt status bit 4 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT3_INT_ST The interrupt status bit 3 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT2_INT_ST The interrupt status bit 2 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT1_INT_ST The interrupt status bit 1 for Host to interrupt Slave. (RO)

SLC0INT_SLC_FRHOST_BIT0_INT_ST The interrupt status bit 0 for Host to interrupt Slave. (RO)

Register 8.4: SLC0INT_ENA_REG (0xC)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_ENA	SLC0INT_SLC0_TX_DSCR_ERR_INT_ENA	(reserved)	SLC0INT_SLC0_RX_EOF_INT_ENA	SLC0INT_SLC0_RX_DONE_INT_ENA	SLC0INT_SLC0_TX_SUC_EOF_INT_ENA	SLC0INT_SLC0_TX_DONE_INT_ENA	(reserved)	SLC0INT_SLC0_TX_OVF_INT_ENA	SLC0INT_SLC0_RX_UDF_INT_ENA	SLC0INT_SLC0_TX_START_INT_ENA	SLC0INT_SLC0_RX_START_INT_ENA	SLC0INT_SLC_FRHOST_BIT7_INT_ENA	SLC0INT_SLC_FRHOST_BIT6_INT_ENA	SLC0INT_SLC_FRHOST_BIT5_INT_ENA	SLC0INT_SLC_FRHOST_BIT4_INT_ENA	SLC0INT_SLC_FRHOST_BIT3_INT_ENA	SLC0INT_SLC_FRHOST_BIT2_INT_ENA	SLC0INT_SLC_FRHOST_BIT1_INT_ENA	SLC0INT_SLC_FRHOST_BIT0_INT_ENA	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SLC0INT_SLC0_RX_DSCR_ERR_INT_ENA The interrupt enable bit for Slave sending linked list descriptor error. (R/W)

SLC0INT_SLC0_TX_DSCR_ERR_INT_ENA The interrupt enable bit for Slave receiving linked list descriptor error. (R/W)

SLC0INT_SLC0_RX_EOF_INT_ENA The interrupt enable bit for Slave sending operation completion. (R/W)

SLC0INT_SLC0_RX_DONE_INT_ENA The interrupt enable bit for single buffer's sent interrupt, in Slave sending mode. (R/W)

SLC0INT_SLC0_TX_SUC_EOF_INT_ENA The interrupt enable bit for Slave receiving operation completion. (R/W)

SLC0INT_SLC0_TX_DONE_INT_ENA The interrupt enable bit for single buffer's full event, in Slave receiving mode. (R/W)

SLC0INT_SLC0_TX_OVF_INT_ENA The interrupt enable bit for Slave receiving buffer overflow. (R/W)

SLC0INT_SLC0_RX_UDF_INT_ENA The interrupt enable bit for Slave sending buffer underflow. (R/W)

SLC0INT_SLC0_TX_START_INT_ENA The interrupt enable bit for Slave receiving operation initialization. (R/W)

SLC0INT_SLC0_RX_START_INT_ENA The interrupt enable bit for Slave sending operation initialization. (R/W)

SLC0INT_SLC_FRHOST_BIT7_INT_ENA The interrupt enable bit 7 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT6_INT_ENA The interrupt enable bit 6 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT5_INT_ENA The interrupt enable bit 5 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT4_INT_ENA The interrupt enable bit 4 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT3_INT_ENA The interrupt enable bit 3 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT2_INT_ENA The interrupt enable bit 2 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT1_INT_ENA The interrupt enable bit 1 for Host to interrupt Slave. (R/W)

SLC0INT_SLC_FRHOST_BIT0_INT_ENA The interrupt enable bit 0 for Host to interrupt Slave. (R/W)

Register 8.5: SLC0INT_CLR_REG (0x10)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_CLR	SLC0INT_SLC0_TX_DSCR_ERR_INT_CLR	(reserved)	SLC0INT_SLC0_RX_EOF_INT_CLR	SLC0INT_SLC0_RX_DONE_INT_CLR	SLC0INT_SLC0_TX_SUC_EOF_INT_CLR	SLC0INT_SLC0_TX_DONE_INT_CLR	(reserved)	SLC0INT_SLC0_TX_OVF_INT_CLR	SLC0INT_SLC0_RX_UDF_INT_CLR	SLC0INT_SLC0_TX_START_INT_CLR	SLC0INT_SLC0_RX_START_INT_CLR	SLC0INT_SLC_FRHOST_BIT7_INT_CLR	SLC0INT_SLC_FRHOST_BIT6_INT_CLR	SLC0INT_SLC_FRHOST_BIT5_INT_CLR	SLC0INT_SLC_FRHOST_BIT4_INT_CLR	SLC0INT_SLC_FRHOST_BIT3_INT_CLR	SLC0INT_SLC_FRHOST_BIT2_INT_CLR	SLC0INT_SLC_FRHOST_BIT1_INT_CLR	SLC0INT_SLC_FRHOST_BIT0_INT_CLR	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 0 0 0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SLC0INT_SLC0_RX_DSCR_ERR_INT_CLR Interrupt clear bit for Slave sending linked list descriptor error. (WO)

SLC0INT_SLC0_TX_DSCR_ERR_INT_CLR Interrupt clear bit for Slave receiving linked list descriptor error. (WO)

SLC0INT_SLC0_RX_EOF_INT_CLR Interrupt clear bit for Slave sending operation completion. (WO)

SLC0INT_SLC0_RX_DONE_INT_CLR Interrupt clear bit for single buffer's sent interrupt, in Slave sending mode. (WO)

SLC0INT_SLC0_TX_SUC_EOF_INT_CLR Interrupt clear bit for Slave receiving operation completion. (WO)

SLC0INT_SLC0_TX_DONE_INT_CLR Interrupt clear bit for single buffer's full event, in Slave receiving mode. (WO)

SLC0INT_SLC0_TX_OVF_INT_CLR Set this bit to clear the Slave receiving overflow interrupt. (WO)

SLC0INT_SLC0_RX_UDF_INT_CLR Set this bit to clear the Slave sending underflow interrupt. (WO)

SLC0INT_SLC0_TX_START_INT_CLR Set this bit to clear the interrupt for Slave receiving operation initialization. (WO)

SLC0INT_SLC0_RX_START_INT_CLR Set this bit to clear the interrupt for Slave sending operation initialization. (WO)

SLC0INT_SLC_FRHOST_BIT7_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT7_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT6_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT6_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT5_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT5_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT4_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT4_INT interrupt. (WO)

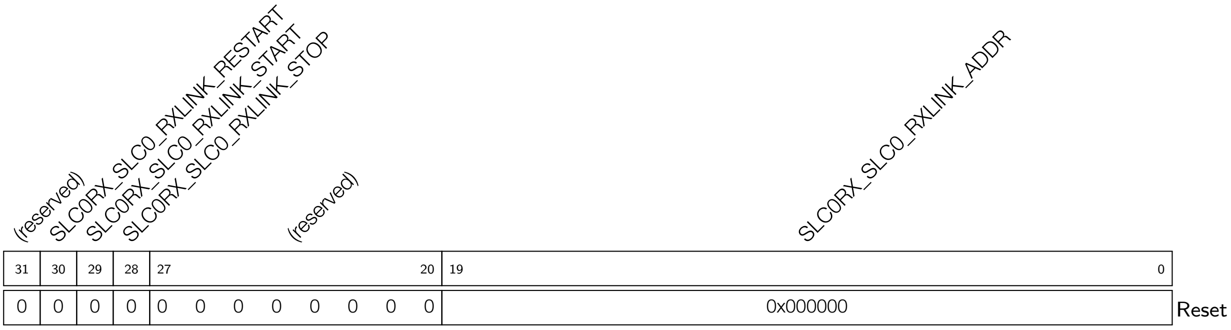
SLC0INT_SLC_FRHOST_BIT3_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT3_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT2_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT2_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT1_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT1_INT interrupt. (WO)

SLC0INT_SLC_FRHOST_BIT0_INT_CLR Set this bit to clear the SLC0INT_SLC_FRHOST_BIT0_INT interrupt. (WO)

Register 8.6: SLC0RX_LINK_REG (0x3C)



31	30	29	28	27 … 20	19 … 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

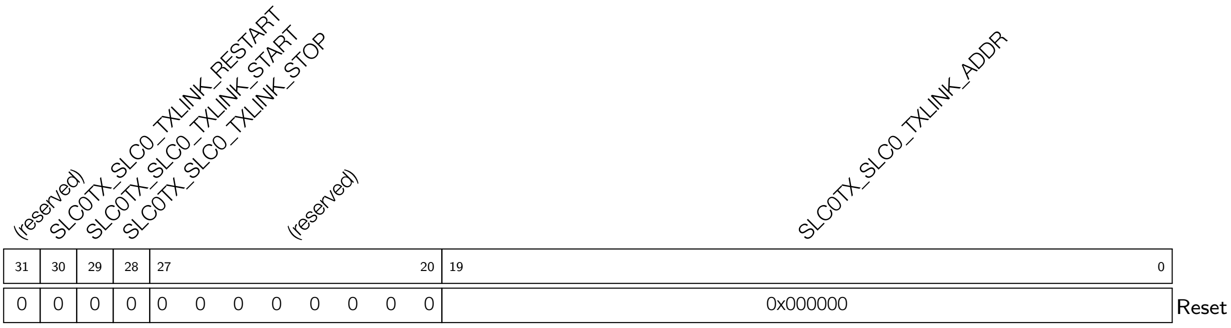
SLC0RX_SLC0_RXLINK_RESTART Set this bit to restart and continue the linked list operation for sending packets. (R/W)

SLC0RX_SLC0_RXLINK_START Set this bit to start the linked list operation for sending packets. Sending will start from the address indicated by SLC0_RXLINK_ADDR. (R/W)

SLC0RX_SLC0_RXLINK_STOP Set this bit to stop the linked list operation. (R/W)

SLC0RX_SLC0_RXLINK_ADDR The lowest 20 bits in the initial address of Slave's sending linked list. (R/W)

Register 8.7: SLC0TX_LINK_REG (0x40)



31	30	29	28	27 … 20	19 … 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

SLC0TX_SLC0_TXLINK_RESTART Set this bit to restart and continue the linked list operation for receiving packets. (R/W)

SLC0TX_SLC0_TXLINK_START Set this bit to start the linked list operation for receiving packets. Receiving will start from the address indicated by SLC0_TXLINK_ADDR. (R/W)

SLC0TX_SLC0_TXLINK_STOP Set this bit to stop the linked list operation for receiving packets. (R/W)

SLC0TX_SLC0_TXLINK_ADDR The lowest 20 bits in the initial address of Slave's receiving linked list. (R/W)

Register 8.8: SLCINTVEC_TOHOST_REG (0x4C)

(reserved)	(reserved)	(reserved)	SLCINTVEC_SLC0_TOHOST_INTVEC	
31 24	23 16	15 8	7 0	
0x000	0 0 0 0 0 0 0 0	0x000	0x000	Reset

SLCINTVEC_SLC0_TOHOST_INTVEC The interrupt vector for Slave to interrupt Host. (WO)

Register 8.9: SLC0TOKEN1_REG (0x54)

(reserved)	SLC0TOKEN1_SLC0_TOKEN1	(reserved)	SLC0TOKEN1_SLC0_TOKEN1_INC_MORE	(reserved)	SLC0TOKEN1_SLC0_TOKEN1_WDATA	
31 28	27 16	15	14	13 12	11 0	
0x00	0x0000	0	0	0 0	0x0000	Reset

SLC0TOKEN1_SLC0_TOKEN1 The accumulated number of buffers for receiving packets. (RO)

SLC0TOKEN1_SLC0_TOKEN1_INC_MORE Set this bit to add the value of SLC0TOKEN1_SLC0_TOKEN1_WDATA to that of SLC0TOKEN1_SLC0_TOKEN1. (WO)

SLC0TOKEN1_SLC0_TOKEN1_WDATA The number of available receiving buffers. (WO)

Register 8.10: SLCCONF1_REG (0x60)

(reserved)	(reserved)	(reserved)	SLCCONF1_SLC0_RX_STITCH_EN	SLCCONF1_SLC0_TX_STITCH_EN	SLCCONF1_SLC0_LEN_AUTO_CLR	
31 23	22 16	15 7	6	5	4	
0x000	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	1	1	1	Reset

SLCCONF1_SLC0_RX_STITCH_EN Please initialize to 0. Do not modify it. (R/W)

SLCCONF1_SLC0_TX_STITCH_EN Please initialize to 0. Do not modify it. (R/W)

SLCCONF1_SLC0_LEN_AUTO_CLR Please initialize to 0. Do not modify it. (R/W)

Register 8.11: SLC_RX_DSCR_CONF_REG (0x98)

(reserved)	SLC_SLC0_TOKEN_NO_REPLACE	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

SLC_SLC0_TOKEN_NO_REPLACE Please initialize to 1. Do not modify it. (R/W)

Register 8.12: SLC0_LEN_CONF_REG (0xE4)

(reserved)	(reserved)	SLC0_LEN_INC_MORE	(reserved)	SLC0_LEN_WDATA	
31 29	28 23	22	21 20	19 0	
0x0	0 0 0 0 0 0	0	0 0	0x000000	Reset

SLC0_LEN_INC_MORE Set this bit to add the value of SLC0_LEN to that of SLC0_LEN_WDATA. (WO)

SLC0_LEN_WDATA The packet length sent. (WO)

Register 8.13: SLC0_LENGTH_REG (0xE8)

(reserved)		SLC0_LEN		
31	20	19	0	
0x0000		0x000000		Reset

SLC0_LEN Indicates the packet length sent by the Slave. (RO)

8.6 SLC Host Registers

The second block of SDIO control registers starts at 0x3FF5_5000.

Register 8.14: SLC0HOST_TOKEN_RDATA (0x44)

(reserved)		HOSTREG_SLC0_TOKEN1		(reserved)		
31	28	27	16	15	0	
0x000		0x000		0x000		Reset

HOSTREG_SLC0_TOKEN1 The accumulated number of Slave's receiving buffers. (RO)

Register 8.15: SLC0HOST_INT_RAW_REG (0x50)

31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_RAW	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_RAW	SLC0HOST_SLC0_RX_UDF_INT_RAW	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT6_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT5_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT4_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT3_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT2_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT1_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT0_INT_	
0x00	0 0	0	0 0 0 0 0	0	0	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

SLC0HOST_SLC0_RX_NEW_PACKET_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_RX_NEW_PACKET_INT interrupt. (RO)

SLC0HOST_SLC0_TX_OVF_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TX_OVF_INT interrupt. (RO)

SLC0HOST_SLC0_RX_UDF_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_RX_UDF_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT7_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT7_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT6_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT6_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT5_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT5_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT4_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT4_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT3_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT3_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT2_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT2_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT1_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT1_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT0_INT_RAW The raw interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT0_INT interrupt. (RO)

Register 8.16: SLC0HOST_INT_ST_REG (0x58)

(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_ST	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_ST	SLC0HOST_SLC0_RX_UDF_INT_ST	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_ST	SLC0HOST_SLC0_TOHOST_BIT6_INT_ST	SLC0HOST_SLC0_TOHOST_BIT5_INT_ST	SLC0HOST_SLC0_TOHOST_BIT4_INT_ST	SLC0HOST_SLC0_TOHOST_BIT3_INT_ST	SLC0HOST_SLC0_TOHOST_BIT2_INT_ST	SLC0HOST_SLC0_TOHOST_BIT1_INT_ST	SLC0HOST_SLC0_TOHOST_BIT0_INT_	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	0	0 0 0 0 0	0	0	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

SLC0HOST_SLC0_RX_NEW_PACKET_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_RX_NEW_PACKET_INT interrupt. (RO)

SLC0HOST_SLC0_TX_OVF_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TX_OVF_INT interrupt. (RO)

SLC0HOST_SLC0_RX_UDF_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_RX_UDF_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT7_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT7_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT6_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT6_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT5_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT5_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT4_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT4_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT3_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT3_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT2_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT2_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT1_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT1_INT interrupt. (RO)

SLC0HOST_SLC0_TOHOST_BIT0_INT_ST The masked interrupt status bit for the SLC0HOST_SLC0_TOHOST_BIT0_INT interrupt. (RO)

Register 8.17: SLCHOST_PKT_LEN_REG (0x60)

SLCHOST_HOSTREG_SLC0_LEN_CHECK	SLCHOST_HOSTREG_SLC0_LEN	
31 20	19 0	
0x000	0x000	Reset

SLCHOST_HOSTREG_SLC0_LEN_CHECK Its value is HOSTREG_SLC0_LEN[9:0] plus HOSTREG_SLC0_LEN[19:10]. (RO)

SLCHOST_HOSTREG_SLC0_LEN The accumulated value of the data length sent by the Slave. The value gets updated only when the Host reads it.

Register 8.18: SLCHOST_CONF_W0_REG (0x6C)

SLCHOST_CONF3	SLCHOST_CONF2	SLCHOST_CONF1	SLCHOST_CONF0	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF3 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF2 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF1 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF0 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.19: SLCHOST_CONF_W1_REG (0x70)

SLCHOST_CONF7	SLCHOST_CONF6	SLCHOST_CONF5	SLCHOST_CONF4	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF7 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF6 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF5 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF4 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.20: SLCHOST_CONF_W2_REG (0x74)

SLCHOST_CONF11	SLCHOST_CONF10	SLCHOST_CONF9	SLCHOST_CONF8	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF11 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF10 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF9 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF8 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.21: SLCHOST_CONF_W3_REG (0x78)

SLCHOST_CONF15	SLCHOST_CONF14	
31 24	23 16	
0x000	0x000	Reset

SLCHOST_CONF15 The information interaction register between Host and Slave. Both Host and Slave can be read from and written to this. (R/W)

SLCHOST_CONF14 The information interaction register between Host and Slave. Both Host and Slave can be read from and written to this. (R/W)

Register 8.22: SLCHOST_CONF_W4_REG (0x7C)

SLCHOST_CONF19	SLCHOST_CONF18	
31 24	23 16	
0x000	0x000	Reset

SLCHOST_CONF19 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF18 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.23: SLCHOST_CONF_W6_REG (0x88)

SLCHOST_CONF27	SLCHOST_CONF26	SLCHOST_CONF25	SLCHOST_CONF24	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF27 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF26 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF25 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF24 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.24: SLCHOST_CONF_W7_REG (0x8C)

SLCHOST_CONF31	(reserved)	SLCHOST_CONF29	(reserved)	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0x000	0 0 0 0 0 0 0 0	0x000	Reset

SLCHOST_CONF31 The interrupt vector used by Host to interrupt Slave. This bit will not be cleared automatically. (WO)

SLCHOST_CONF29 The interrupt vector used by Host to interrupt Slave. This bit will not be cleared automatically. (WO)

Register 8.25: SLCHOST_CONF_W8_REG (0x9C)

SLCHOST_CONF35	SLCHOST_CONF34	SLCHOST_CONF33	SLCHOST_CONF32	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF35 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF34 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF33 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF32 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.26: SLCHOST_CONF_W9_REG (0xA0)

SLCHOST_CONF39	SLCHOST_CONF38	SLCHOST_CONF37	SLCHOST_CONF36	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

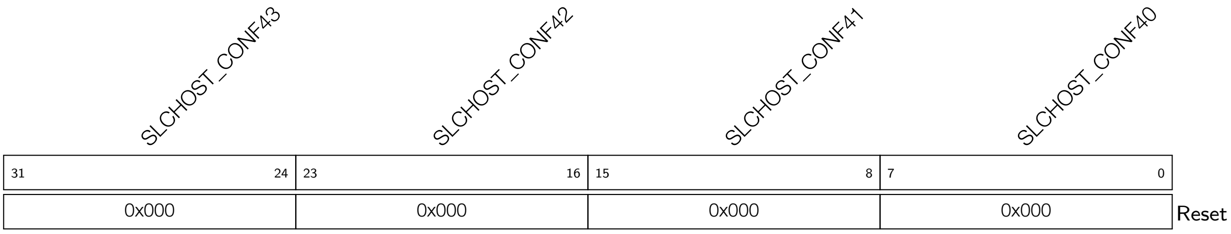
SLCHOST_CONF39 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF38 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF37 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF36 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.27: SLCHOST_CONF_W10_REG (0xA4)



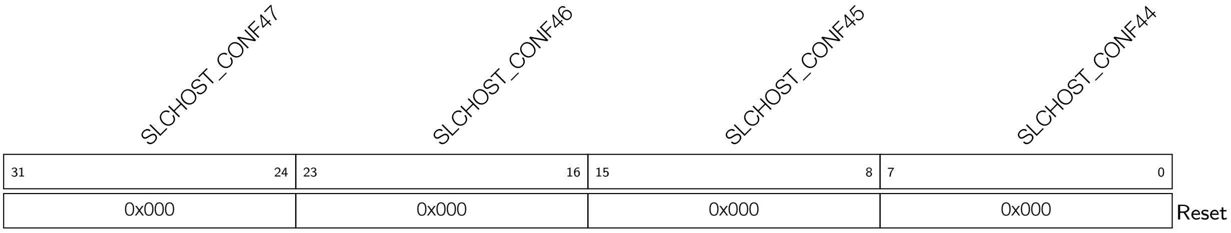
SLCHOST_CONF43 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF42 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF41 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF40 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.28: SLCHOST_CONF_W11_REG (0xA8)



SLCHOST_CONF47 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF46 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF45 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF44 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.29: SLCHOST_CONF_W12_REG (0xAC)

SLCHOST_CONF51	SLCHOST_CONF50	SLCHOST_CONF49	SLCHOST_CONF48	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF51 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF50 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF49 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF48 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.30: SLCHOST_CONF_W13_REG (0xB0)

SLCHOST_CONF55	SLCHOST_CONF54	SLCHOST_CONF53	SLCHOST_CONF52	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF55 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF54 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF53 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF52 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.31: SLCHOST_CONF_W14_REG (0xB4)

SLCHOST_CONF59	SLCHOST_CONF58	SLCHOST_CONF57	SLCHOST_CONF56	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF59 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF58 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF57 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF56 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.32: SLCHOST_CONF_W15_REG (0xB8)

SLCHOST_CONF63	SLCHOST_CONF62	SLCHOST_CONF61	SLCHOST_CONF60	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF63 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF62 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF61 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

SLCHOST_CONF60 The information interaction register between Host and Slave. Both Host and Slave can access it. (R/W)

Register 8.33: SLC0HOST_INT_CLR_REG (0xD4)

(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_CLR	SLC0HOST_SLC0_RX_UDF_INT_CLR	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT0_INT_	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	0	0 0 0 0 0	0	0	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR Set this bit to clear the SLC0HOST_SLC0_RX_NEW_PACKET_INT interrupt. (WO)

SLC0HOST_SLC0_TX_OVF_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TX_OVF_INT interrupt. (WO)

SLC0HOST_SLC0_RX_UDF_INT_CLR Set this bit to clear the SLC0HOST_SLC0_RX_UDF_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT7_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT6_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT5_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT4_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT3_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT2_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT1_INT interrupt. (WO)

SLC0HOST_SLC0_TOHOST_BIT0_INT_CLR Set this bit to clear the SLC0HOST_SLC0_TOHOST_BIT0_INT interrupt. (WO)

Register 8.34: SLC0HOST_FUNC1_INT_ENA_REG (0xDC)

(reserved)	(reserved)	SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA	(reserved)	SLC0HOST_FN1_SLC0_TX_OVF_INT_ENA	SLC0HOST_FN1_SLC0_RX_UDF_INT_ENA	(reserved)	SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT_E	SLC0HOST_FN1_SLC0_TOHOST_BIT0	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	0	0 0 0 0 0	0	0	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TX_OVF_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TX_OVF_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_RX_UDF_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_RX_UDF_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT interrupt. (R/W)

SLC0HOST_FN1_SLC0_TOHOST_BIT0_INT_ENA The interrupt enable bit for the SLC0HOST_FN1_SLC0_TOHOST_BIT0_INT interrupt. (R/W)

Register 8.35: SLCHOST_CONF_REG (0x1F0)

(reserved)	(reserved)	SLCHOST_FRC_POS_SAMP	SLCHOST_FRC_NEG_SAMP	SLCHOST_FRC_SDIO20	SLCHOST_FRC_SDIO11	
31 28	27 20	19 15	14 10	9 5	4 0	
0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

SLCHOST_FRC_POS_SAMP Set this bit to sample the corresponding signal at the rising clock edge. (R/W)

SLCHOST_FRC_NEG_SAMP Set this bit to sample the corresponding signal at the falling clock edge. (R/W)

SLCHOST_FRC_SDIO20 Set this bit to output the corresponding signal at the rising clock edge. (R/W)

SLCHOST_FRC_SDIO11 Set this bit to output the corresponding signal at the falling clock edge. (R/W)

8.7 HINF Registers

The third block of SDIO control registers starts at 0x3FF4_B000.

Register 8.36: HINF_CFG_DATA1_REG (0x4)

(reserved)	HINF_HIGHSPEED_ENABLE	HINF_SDIO_IOREADY1	
31 3	2	1	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

HINF_HIGHSPEED_ENABLE Please initialize to 1. Do not modify it. (R/W)

HINF_SDIO_IOREADY1 Please initialize to 1. Do not modify it. (R/W)

9. SD/MMC Host Controller

9.1 Overview

The ESP32 memory card interface controller provides a hardware interface between the Advanced Peripheral Bus (APB) and an external memory device. The memory card interface allows the ESP32 to be connected to SDIO memory cards, MMC cards and devices with a CE-ATA interface. It supports two external cards (Card0 and Card1).

9.2 Features

This module has the following features:

- Two external cards
- Supports SD Memory Card standard: versions 3.0 and 3.01
- Supports MMC: versions 4.41, 4.5, and 4.51
- Supports CE-ATA: version 1.1
- Supports 1-bit, 4-bit, and 8-bit (Card0 only) modes

The SD/MMC controller topology is shown in Figure 31. The controller supports two peripherals which cannot be functional at the same time.

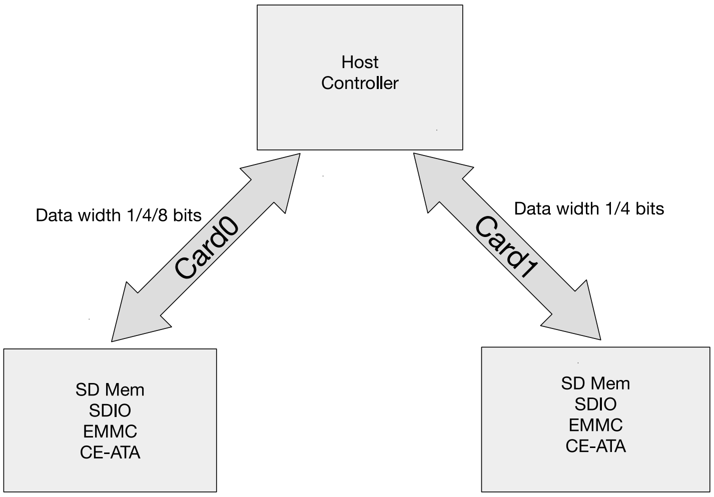


Figure 31: SD/MMC Controller Topology

9.3 SD/MMC External Interface Signals

The primary external interface signals, which enable the SD/MMC controller to communicate with an external device, are clock (clk), command (cmd) and data signals. Additional signals include the card interrupt, card detect, and write-protect signals. The direction of each signal is shown in Figure 32. The direction and description of each pin are listed in Table 32.

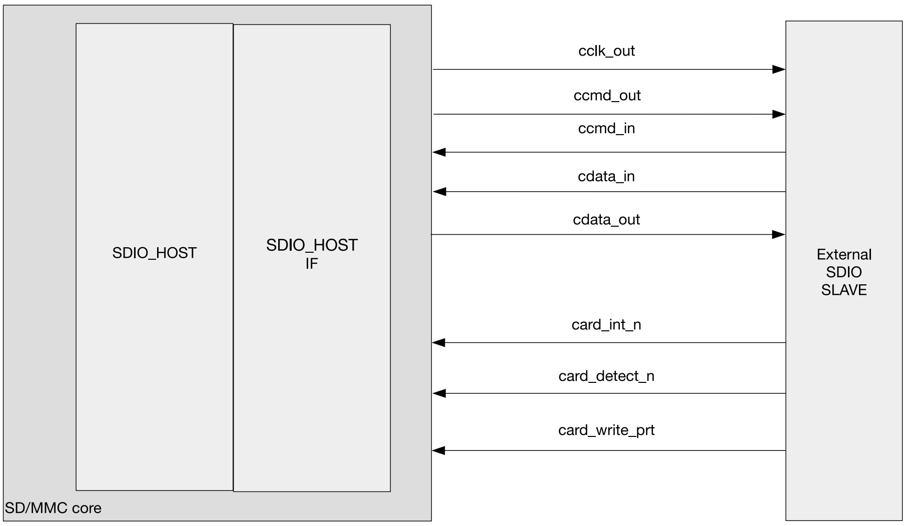


Figure 32: SD/MMC Controller External Interface Signals

Table 32: SD/MMC Signal Description

Pin	Direction	Description
cclk_out	Output	Clock signals for slave device
ccmd	Duplex	Duplex command/response lines
cdata	Duplex	Duplex data read/write lines
card_detect_n	Input	Card detection input line
card_write_prt	Input	Card write protection status input

9.4 Functional Description

9.4.1 SD/MMC Host Controller Architecture

The SD/MMC host controller consists of two main functional blocks, as shown in Figure 33:

- Bus Interface Unit (BIU): It provides APB interfaces for registers, data read and write operation by FIFO and DMA.
- Card Interface Unit (CIU): It handles external memory card interface protocols. It also provides clock control.

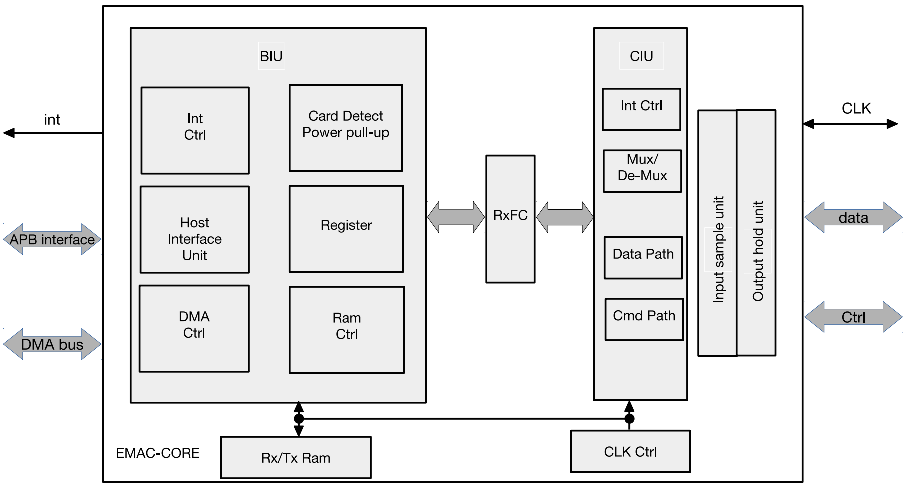


Figure 33: SDIO Host Block Diagram

9.4.1.1 BIU

The BIU provides the access to registers and FIFO data through the Host Interface Unit (HIU). Additionally, it provides FIFO access to independent data through a DMA interface. The host interface can be configured as an APB interface. Figure 33 illustrates the internal components of the BIU. The BIU provides the following functions:

- Host interface
- DMA interface
- Interrupt control
- Register access
- FIFO access
- Power/pull-up control and card detection

9.4.1.2 CIU

The CIU module implements the card-specific protocols. Within the CIU, the command path control unit and data path control unit prompt the controller to interface with the command and data ports, respectively, of the SD/MMC/CE-ATA cards. The CIU also provides clock control. Figure 33 illustrates the internal structure of the CIU, which consists of the following primary functional blocks:

- Command path
- Data path
- SDIO interrupt control
- Clock control
- Mux/demux unit

9.4.2 Command Path

The command path performs the following functions:

- Configures clock parameters
- Configures card command parameters
- Sends commands to card bus (ccmd_out line)
- Receives responses from card bus (ccmd_in line)
- Sends responses to BIU
- Drives the P-bit on the command line

The command path State Machine is shown in Figure 34.

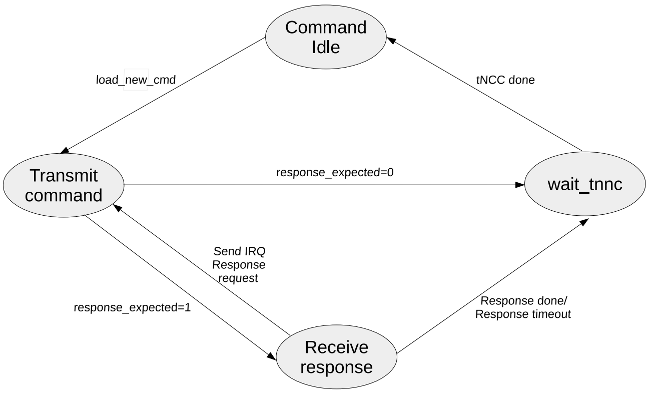


Figure 34: Command Path State Machine

9.4.3 Data Path

The data path block pops FIFO data and transmits them on cdata_out during a write-data transfer, or it receives data on cdata_in and pushes them into FIFO during a read-data transfer. The data path loads new data parameters, i.e., expected data, read/write data transfer, stream/block transfer, block size, byte count, card type, timeout registers, etc., whenever a data transfer command is not in progress.

If the data_expected bit is set in the Command register, the new command is a data-transfer command and the data path starts one of the following operations:

- Transmitting data if the read/write bit = 1
- Receiving data if read/write bit = 0

9.4.3.1 Data Transmit Operation

The data transmit state machine is illustrated in Figure 35. The module starts data transmission two clock cycles after a response for the data-write command is received. This occurs even if the command path detects a response error or a cyclic redundancy check (CRC) error in a response. If no response is received from the card until the response timeout, no data are transmitted. Depending on the value of the transfer_mode bit in the Command register, the data-transmit state machine adds data to the card's data bus in a stream or in block(s). The data transmit state machine is shown in Figure 35.

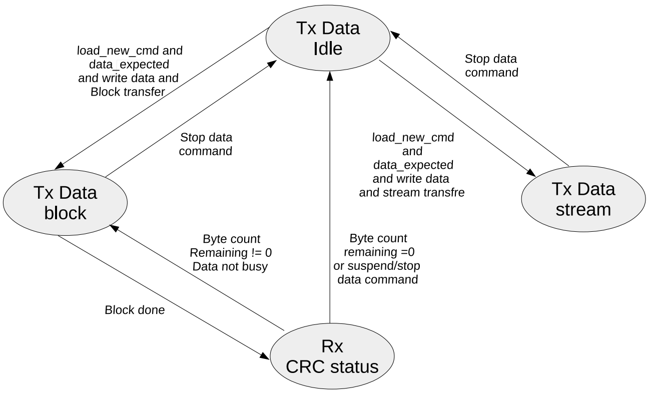

Tx Data Idle
load_new_cmd and data_expected and write data and Block transfer
Stop data command
Stop data command
load_new_cmd and data_expected and write data and stream transfre
Tx Data block
Tx Data stream
Byte count Remaining != 0 Data not busy
Byte count remaining =0 or suspend/stop data command
Block done
Rx CRC status

Figure 35: Data Transmit State Machine

9.4.3.2 Data Receive Operation

The data-receive state machine is illustrated in Figure 36. The module receives data two clock cycles after the end bit of a data-read command, even if the command path detects a response error or a CRC error. If no response is received from the card and a response timeout occurs, the BIU does not receive a signal about the completion of the data transfer. If the command sent by the CIU is an illegal operation for the card, it would prevent the card from starting a read-data transfer, and the BIU will not receive a signal about the completion of the data transfer.

If no data are received by the data timeout, the data path signals a data timeout to the BIU, which marks an end to the data transfer. Based on the value of the transfer_mode bit in the Command register, the data-receive state machine gets data from the card's data bus in a stream or block(s). The data receive state machine is shown in Figure 36.

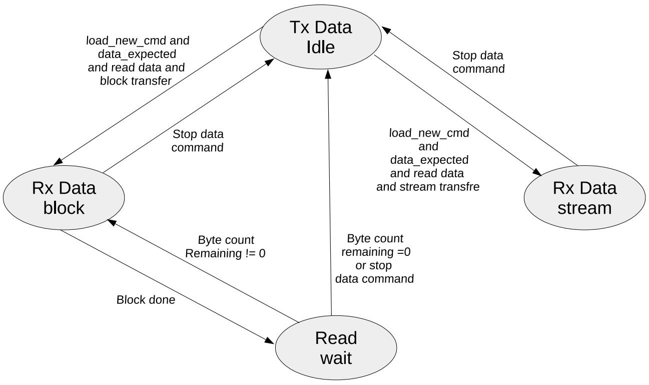

Tx Data Idle
load_new_cmd and data_expected and read data and block transfer
Stop data command
Stop data command
load_new_cmd and data_expected and read data and stream transfre
Rx Data block
Rx Data stream
Byte count Remaining != 0
Byte count remaining =0 or stop data command
Block done
Read wait

Figure 36: Data Receive State Machine

9.5 Software Restrictions for Proper CIU Operation

- Only one card at a time can be selected to execute a command or data transfer. For example, when data are being transferred to or from a card, a new command must not be issued to another card. A new command, however, can be issued to the same card, allowing it to read the device status or stop the transfer.
- Only one command at a time can be issued for data transfers.
- During an open-ended card-write operation, if the card clock is stopped due to FIFO being empty, the software must fill FIFO with data first, and then start the card clock. Only then can it issue a stop/abort command to the card.
- During an SDIO/COMBO card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset FIFO, and then issue the resume command as if it were a new data-transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52_reset), while a card data transfer is in progress, the software must set the stop_abort_cmd bit in the Command register, so that the CIU can stop the data transfer after issuing the card reset command.
- When the data's end bit error is set in the RINTSTS register, the CIU does not guarantee SDIO interrupts. In such a case, the software ignores SDIO interrupts and issues a stop/abort command to the card, so that the card stops sending read-data.

- If the card clock is stopped due to FIFO being full during a card read, the software will read at least two FIFO locations to restart the card clock.
- Only one CE-ATA device at a time can be selected for a command or data transfer. For example, when data are transferred from a CE-ATA device, a new command should not be sent to another CE-ATA device.
- If a CE-ATA device's interrupts are enabled (nIEN=0), a new RW_BLK command should not be sent to the same device if the execution of a RW_BLK command is already in progress (the RW_BLK command used in this databook is the RW_MULTIPLE_BLOCK MMC command defined by the CE-ATA specifications). Only the CCSD can be sent while waiting for the CCS.
- If, however, a CE-ATA device's interrupts are disabled (nIEN=1), a new command can be issued to the same device, allowing it to read status information.
- Open-ended transfers are not supported in CE-ATA devices.
- The send_auto_stop signal is not supported (software should not set the send_auto_stop bit) in CE-ATA transfers.

After configuring the command start bit to 1, the values of the following registers cannot be changed before a command has been issued:

- CMD - command
- CMDARG - command argument
- BYTCNT - byte count
- BLKSIZ - block size
- CLKDIV - clock divider
- CKLENA - clock enable
- CLKSRC - clock source
- TMOUT - timeout
- CTYPE - card type

9.6 RAM for Receiving and Sending Data

The submodule RAM is a buffer area for sending and receiving data. It can be divided into two units: the one is for sending data, and the other is for receiving data. The process of sending and receiving data can also be achieved by the CPU and DMA for reading and writing. The latter method is described in detail in Section 9.8.

9.6.1 Transmit RAM Module

There are two ways to enable a write operation: DMA and CPU read/write.

If SDIO-sending is enabled, data can be written to the transferred RAM module by APB interface or DMA. Data will be written from register EMAC_FIFO to the CPU, directly, by an APB interface.

9.6.2 Receive RAM Module

There are two ways to enable a read operation: DMA and CPU read/write.

When a subunit of the data path receives data, the subdata will be written onto the receive-RAM. Then, these subdata can be read either with the APB or the DMA method at the reading end. Register EMAC_FIFO can be read by the APB directly.

9.7 Descriptor Chain

Each linked list module consists of two parts: the linked list itself and a data buffer. In other words, each module points to a unique data buffer and the linked list that follows the module. Figure 37 shows the descriptor chain.

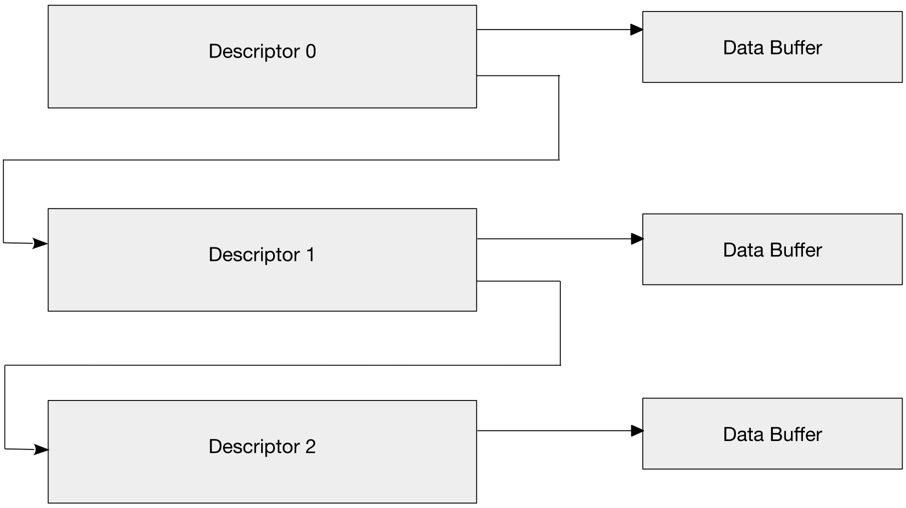


Figure 37: Descriptor Chain

9.8 The Structure of a Linked List

Each linked list consists of four words. As is shown below, Figure 38 demonstrates the linked list's structure, and Table 33, Table 34, Table 35, Table 36 provide the descriptions of linked lists.

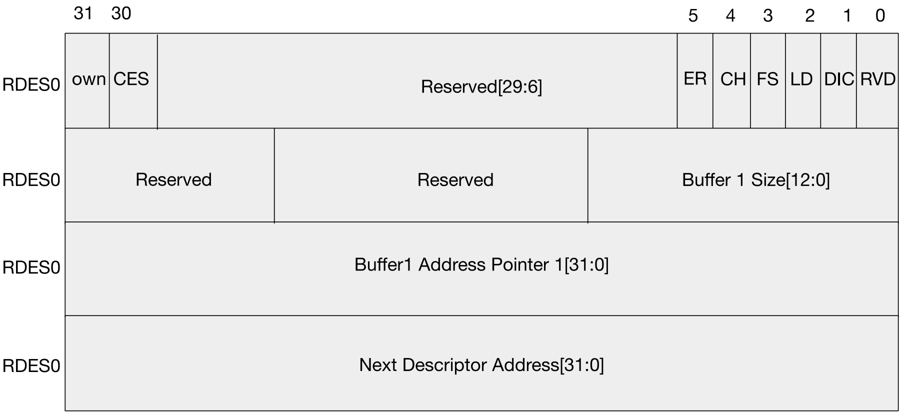

<table>
<tr><th></th><th>31</th><th>30</th><th></th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>RDES0</td><td>own</td><td>CES</td><td>Reserved[29:6]</td><td>ER</td><td>CH</td><td>FS</td><td>LD</td><td>DIC</td><td>RVD</td></tr>
<tr><td>RDES0</td><td colspan="2">Reserved</td><td>Reserved</td><td colspan="6">Buffer 1 Size[12:0]</td></tr>
<tr><td>RDES0</td><td colspan="9">Buffer1 Address Pointer 1[31:0]</td></tr>
<tr><td>RDES0</td><td colspan="9">Next Descriptor Address[31:0]</td></tr>
</table>

Figure 38: The Structure of a Linked List

The DES0 element contains control and status information.

Table 33: DES0

Bits	Name	Description
31	OWN	When set, this bit indicates that the descriptor is owned by the DMAC. When reset, it indicates that the descriptor is owned by the Host. The DMAC clears this bit when it completes the data transfer.
30	CES (Card Error Summary)	These error bits indicate the status of the transition to or from the card. The following bits are also present in RINTSTS, which indicates their digital logic OR gate. • EBE: End Bit Error • RTO: Response Time out • RCRC: Response CRC • SBE: Start Bit Error • DRTO: Data Read Timeout • DCRC: Data CRC for Receive • RE: Response Error
29:6	Reserved	Reserved
5	ER (End of Ring)	When set, this bit indicates that the descriptor list has reached its final descriptor. The DMAC then returns to the base address of the list, creating a Descriptor Ring.
4	CH (Second Address Chained)	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address. When this bit is set, BS2 (DES1[25:13]) should be all zeros.
3	FD (First Descriptor)	When set, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, the Next Descriptor contains the beginning of the data.
2	LD (Last Descriptor)	This bit is associated with the last block of a DMA transfer. When set, the bit indicates that the buffers pointed by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count should be 0.
1	DIC (Disable Interrupt on Completion)	When set, this bit will prevent the setting of the TI/RI bit of the DMAC Status Register (IDSTS) for the data that ends in the buffer pointed by this descriptor.
0	Reserved	Reserved

The DES1 element contains the buffer size.

Table 34: DES1

Bits	Name	Description
31:26	Reserved	Reserved
25:13	Reserved	Reserved
12:0	BS1 (Buffer 1 Size)	Indicates the data buffer byte size, which must be a multiple of four. In the case where the buffer size is not a multiple of four, the resulting behavior is undefined. This field should not be zero.

The DES2 element contains the address pointer to the data buffer.

Table 35: DES2

Bits	Name	Description
31:0	Buffer Address Pointer 1	These bits indicate the physical address of the data buffer.

The DES3 element contains the address pointer to the next descriptor if the present descriptor is not the last one in a chained descriptor structure.

Table 36: DES3

Bits	Name	Description
31:0	Next Descriptor Address	If the Second Address Chained (DES0[4]) bit is set, then this address contains the pointer to the physical memory where the Next Descriptor is present. If this is not the last descriptor, then the Next Descriptor address pointer must be DES3[1:0] = 0.

9.9 Initialization

9.9.1 DMAC Initialization

The DMAC initialization should proceed as follows:

- Write to the DMAC Bus Mode Register (BMOD_REG) will set the Host bus's access parameters.
- Write to the DMAC Interrupt Enable Register (IDINTEN) will mask any unnecessary interrupt causes.
- The software driver creates either the transmit or the receive descriptor list. Then, it writes to the DMAC Descriptor List Base Address Register (DBADDR), providing the DMAC with the starting address of the list.
- The DMAC engine attempts to acquire descriptors from descriptor lists.

9.9.2 DMAC Transmission Initialization

The DMAC transmission occurs as follows:

1. The Host sets up the elements (DES0-DES3) for transmission, and sets the OWN bit (DES0[31]). The Host also prepares the data buffer.
2. The Host programs the write-data command in the CMD register in BIU.
3. The Host also programs the required transmit threshold (TX_WMARK field in FIFOTH register).
4. The DMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMAC enters a suspend-state and asserts the Descriptor Unable interrupt in the IDSTS register. In such a case, the host needs to release the DMAC by writing any value to PLDMND_REG.
5. It then waits for the Command Done (CD) bit and no errors from BIU, which indicates that a transfer can be done.
6. Subsequently, the DMAC engine waits for a DMA interface request (dw_dma_req) from BIU. This request will be generated, based on the programmed transmit-threshold value. For the last bytes of data which cannot be accessed using a burst, single transfers are performed on the AHB Master Interface.
7. The DMAC fetches the transmit data from the data buffer in the Host memory and transfers them to FIFO for transmission to card.
8. When data span across multiple descriptors, the DMAC fetches the next descriptor and extends its operation using the following descriptor. The last descriptor bit indicates whether the data span multiple descriptors or not.
9. When data transmission is complete, the status information is updated in the IDSTS register by setting the Transmit Interrupt, if it has already been enabled. Also, the OWN bit is cleared by the DMAC by performing a write transaction to DES0.

9.9.3 DMAC Reception Initialization

The DMAC reception occurs as follows:

1. The Host sets up the element (DES0-DES3) for reception, and sets the OWN bit (DES0[31]).
2. The Host programs the read-data command in the CMD register in BIU.
3. Then, the Host programs the required level of the receive-threshold (RX_WMARK field in FIFOTH register).
4. The DMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMA enters a suspend-state and asserts the Descriptor Unable interrupt in the IDSTS register. In such a case, the host needs to release the DMAC by writing any value to PLDMND_REG.
5. It then waits for the Command Done (CD) bit and no errors from BIU, which indicates that a transfer can be done.
6. The DMAC engine then waits for a DMA interface request (dw_dma_req) from BIU. This request will be generated, based on the programmed receive-threshold value. For the last bytes of the data which cannot be accessed using a burst, single transfers are performed on the AHB.
7. The DMAC fetches the data from FIFO and transfers them to the Host memory.

8. When data span across multiple descriptors, the DMAC will fetch the next descriptor and extend its operation using the following descriptor. The last descriptor bit indicates whether the data span multiple descriptors or not.
9. When data reception is complete, the status information is updated in the IDSTS register by setting Receive-Interrupt, if it has already been enabled. Also, the OWN bit is cleared by the DMAC by performing a write-transaction to DES0.

9.10 Clock Phase Selection

If the setup time requirements for the input or output data signal are not met, users can specify the clock phase, as shown in the figure below.

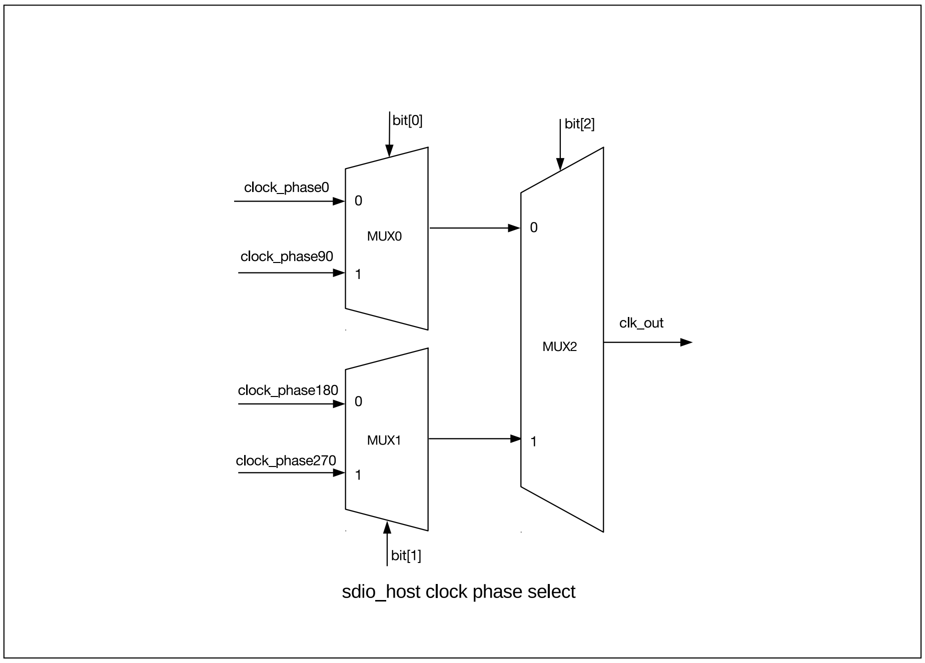


Figure 39: Clock Phase Selection

Please find detailed information on the clock phase selection register CLK_EDGE_SEL in Section Registers.

9.11 Interrupt

Interrupts can be generated as a result of various events. The IDSTS register contains all the bits that might cause an interrupt. The IDINTEN register contains an enable bit for each of the events that can cause an interrupt.

There are two groups of summary interrupts, ”Normal” ones (bit8 NIS) and ”Abnormal” ones (bit9 AIS), as outlined in the IDSTS register. Interrupts are cleared by writing 1 to the position of the corresponding bit. When all the enabled interrupts within a group are cleared, the corresponding summary bit is also cleared. When both summary bits are cleared, the interrupt signal dmac_intr_o is de-asserted (stops signalling).

Interrupts are not queued up, and if a new interrupt-event occurs before the driver has responded to it, no additional interrupts are generated. For example, the Receive Interrupt IDSTS[1] indicates that one or more data were transferred to the Host buffer.

An interrupt is generated only once for concurrent events. The driver must scan the IDSTS register for the interrupt cause.

9.12 Register Summary

Name	Description	Address	Access
CTRL_REG	Control register	0x0000	R/W
CLKDIV_REG	Clock divider configuration register	0x0008	R/W
CLKSRC_REG	Clock source selection register	0x000C	R/W
CLKENA_REG	Clock enable register	0x0010	R/W
TMOUT_REG	Data and response timeout configuration register	0x0014	R/W
CTYPE_REG	Card bus width configuration register	0x0018	R/W
BLKSIZ_REG	Card data block size configuration register	0x001C	R/W
BYTCNT_REG	Data transfer length configuration register	0x0020	R/W
INTMASK_REG	SDIO interrupt mask register	0x0024	R/W
CMDARG_REG	Command argument data register	0x0028	R/W
CMD_REG	Command and boot configuration register	0x002C	R/W
RESP0_REG	Response data register	0x0030	RO
RESP1_REG	Long response data register	0x0034	RO
RESP2_REG	Long response data register	0x0038	RO
RESP3_REG	Long response data register	0x003C	RO
MINTSTS_REG	Masked interrupt status register	0x0040	RO
RINTSTS_REG	Raw interrupt status register	0x0044	R/W
STATUS_REG	SD/MMC status register	0x0048	RO
FIFOTH_REG	FIFO configuration register	0x004C	R/W
CDETECT_REG	Card detect register	0x0050	RO
WRTPRT_REG	Card write protection (WP) status register	0x0054	RO
TCBCNT_REG	Transferred byte count register	0x005C	RO
TBBCNT_REG	Transferred byte count register	0x0060	RO
DEBNCE_REG	Debounce filter time configuration register	0x0064	R/W
USRID_REG	User ID (scratchpad) register	0x0068	R/W
RST_N_REG	Card reset register	0x0078	R/W
BMOD_REG	Burst mode transfer configuration register	0x0080	R/W
PLDMND_REG	Poll demand configuration register	0x0084	WO
DBADDR_REG	Descriptor base address register	0x0088	R/W
IDSTS_REG	IDMAC status register	0x008C	R/W
IDINTEN_REG	IDMAC interrupt enable register	0x0090	R/W
DSCADDR_REG	Host descriptor address pointer	0x0094	RO
BUFADDR_REG	Host buffer address pointer register	0x0098	RO
CLK_EDGE_SEL	Clock phase selection register	0x0800	R/W

9.13 Registers

SD/MMC controller registers can be accessed by the APB bus of the CPU.

Register 9.1: CTRL_REG (0x0000)

(reserved)	(reserved)	(reserved)	CEATA_DEVICE_INTERRUPT_STATUS	SEND_AUTO_STOP_CCSD	SEND_CCSD	ABORT_READ_DATA	SEND_IRQ_RESPONSE	READ_WAIT	(reserved)	INT_ENABLE	(reserved)	DMA_RESET	FIFO_RESET	CONTROLLER_RESET	
31 25	24	131 120	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	1	0x00	0	0	0	0	0	0	0	0	0	0	0	0	Reset

CEATA_DEVICE_INTERRUPT_STATUS Software should appropriately write to this bit after the power-on reset or any other reset to the CE-ATA device. After reset, the CE-ATA device's interrupt is usually disabled (nIEN = 1). If the host enables the CE-ATA device's interrupt, then software should set this bit. (R/W)

SEND_AUTO_STOP_CCSD Always set send_auto_stop_ccsd and send_ccsd bits together; send_auto_stop_ccsd should not be set independently of send_ccsd. When set, SD/MMC automatically sends an internally-generated STOP command (CMD12) to the CE-ATA device. After sending this internally-generated STOP command, the Auto Command Done (ACD) bit in RINTSTS is set and an interrupt is generated for the host, in case the ACD interrupt is not masked. After sending the Command Completion Signal Disable (CCSD), SD/MMC automatically clears the send_auto_stop_ccsd bit. (R/W)

SEND_CCSD When set, SD/MMC sends CCSD to the CE-ATA device. Software sets this bit only if the current command is expecting CCS (that is, RW_BLK), and if interrupts are enabled for the CE-ATA device. Once the CCSD pattern is sent to the device, SD/MMC automatically clears the send_ccsd bit. It also sets the Command Done (CD) bit in the RINTSTS register, and generates an interrupt for the host, in case the Command Done interrupt is not masked. NOTE: Once the send_ccsd bit is set, it takes two card clock cycles to drive the CCSD on the CMD line. Due to this, within the boundary conditions the CCSD may be sent to the CE-ATA device, even if the device has signalled CCS. (R/W)

ABORT_READ_DATA After a suspend-command is issued during a read-operation, software polls the card to find when the suspend-event occurred. Once the suspend-event has occurred, software sets the bit which will reset the data state machine that is waiting for the next block of data. This bit is automatically cleared once the data state machine is reset to idle. (R/W)

SEND_IRQ_RESPONSE Bit automatically clears once response is sent. To wait for MMC card interrupts, host issues CMD40 and waits for interrupt response from MMC card(s). In the meantime, if host wants SD/MMC to exit waiting for interrupt state, it can set this bit, at which time SD/MMC command state-machine sends CMD40 response on bus and returns to idle state. (R/W)

Register 9.2: CTRL_REG (continued) (0x0000)

(reserved)	(reserved)	(reserved)	CEATA_DEVICE_INTERRUPT_STATUS	SEND_AUTO_STOP_CCSD	SEND_CCSD	ABORT_READ_DATA	SEND_IRQ_RESPONSE	READ_WAIT	DMA_ENABLE	INT_ENABLE	(reserved)	DMA_RESET	FIFO_RESET	CONTROLLER_RESET	
31 25	24	131 120	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	1	0x00	0	0	0	0	0	0	0	0	0	0	0	0	Reset

READ_WAIT For sending read-wait to SDIO cards. (R/W)

INT_ENABLE Global interrupt enable/disable bit. 0: Disable; 1: Enable. (R/W)

DMA_RESET To reset DMA interface, firmware should set bit to 1. This bit is auto-cleared after two AHB clocks. (R/W)

FIFO_RESET To reset FIFO, firmware should set bit to 1. This bit is auto-cleared after completion of reset operation. Note: FIFO pointers will be out of reset after 2 cycles of system clocks in addition to synchronization delay (2 cycles of card clock), after the fifo_reset is cleared. (R/W)

CONTROLLER_RESET To reset controller, firmware should set this bit. This bit is auto-cleared after two AHB and two cclk_in clock cycles. (R/W)

Register 9.3: CLKDIV_REG (0x0008)

CLK_DIVIDER3	CLK_DIVIDER2	CLK_DIVIDER1	CLK_DIVIDER0	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

CLK_DIVIDER3 Clock divider-3 value. Clock division factor is 2**n*, where *n*=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

CLK_DIVIDER2 Clock divider-2 value. Clock division factor is 2**n*, where *n*=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

CLK_DIVIDER1 Clock divider-1 value. Clock division factor is 2**n*, where *n*=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

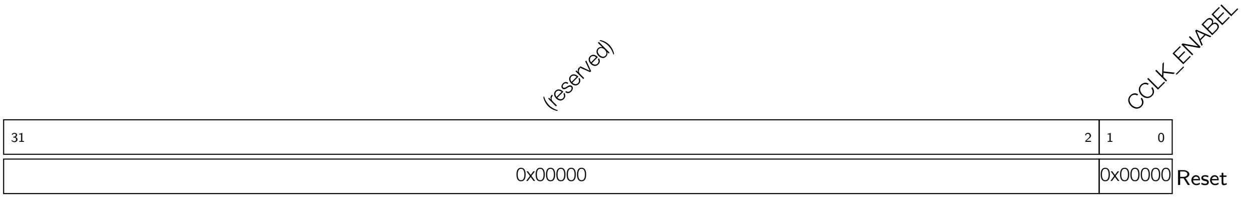
CLK_DIVIDER0 Clock divider-0 value. Clock division factor is 2**n*, where *n*=0 bypasses the divider (division factor of 1). For example, a value of 1 means divide by 2*1 = 2, a value of 0xFF means divide by 2*255 = 510, and so on. In MMC-Ver3.3-only mode, these bits are not implemented because only one clock divider is supported. (R/W)

Register 9.4: CLKSRC_REG (0x000C)

(reserved)	CLKSRC_REG	
31 4	3 0	
0x000000	0x0	Reset

CLKSRC_REG Clock divider source for two SD cards is supported. Each card has two bits assigned to it. For example, bit[1:0] are assigned for card 0, bit[3:2] are assigned for card 1. Card 0 maps and internally routes clock divider[0:3] outputs to cclk_out[1:0] pins, depending on bit value.
00 : Clock divider 0;
01 : Clock divider 1;
10 : Clock divider 2;
11 : Clock divider 3.
In MMC-Ver3.3-only controller, only one clock divider is supported. The cclk_out is always from clock divider 0, and this register is not implemented. (R/W)

Register 9.5: CLKENA_REG (0x0010)



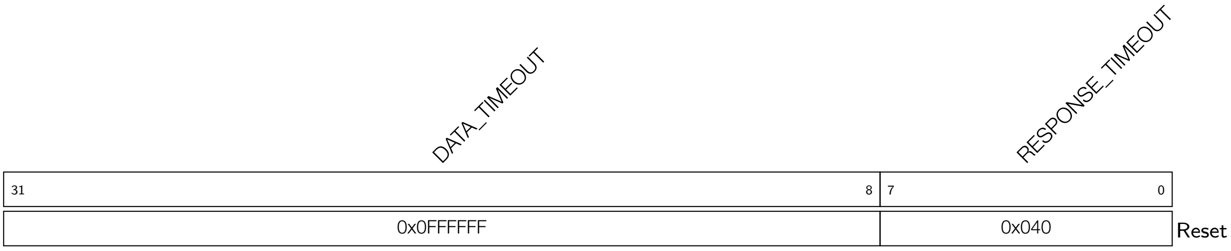
CCLK_ENABEL Clock-enable control for two SD card clocks and one MMC card clock is supported.

0: Clock disabled;

1: Clock enabled.

In MMC-Ver3.3-only mode, since there is only one cclk_out, only cclk_enable[0] is used. (R/W)

Register 9.6: TMOUT_REG (0x0014)



DATA_TIMEOUT Value for card data read timeout. This value is also used for data starvation by host timeout. The timeout counter is started only after the card clock is stopped. This value is specified in number of card output clocks, i.e. cclk_out of the selected card.

NOTE: The software timer should be used if the timeout value is in the order of 100 ms. In this case, read data timeout interrupt needs to be disabled. (R/W)

RESPONSE_TIMEOUT Response timeout value. Value is specified in terms of number of card output clocks, i.e., cclk_out. (R/W)

Register 9.7: CTYPE_REG (0x0018)

(reserved)	CARD_WIDTH8	(reserved)	CARD_WIDTH4	
31 18	17 16	15 2	1 0	
0x00000	0x00000	0x00000	0x00000	Reset

CARD_WIDTH8 One bit per card indicates if card is in 8-bit mode.

0: Non 8-bit mode;

1: 8-bit mode.

Bit[17:16] correspond to card[1:0] respectively. (R/W)

CARD_WIDTH4 One bit per card indicates if card is 1-bit or 4-bit mode.

0: 1-bit mode;

1: 4-bit mode.

Bit[1:0] correspond to card[1:0] respectively. Only NUM_CARDS*2 number of bits are implemented. (R/W)

Register 9.8: BLKSIZ_REG (0x001C)

(reserved)	BLOCK_SIZE	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00200	Reset

BLOCK_SIZE Block size. (R/W)

Register 9.9: BYTCNT_REG (0x0020)

31 0	
0x000000200	Reset

BYTCNT_REG Number of bytes to be transferred, should be an integral multiple of Block Size for block transfers. For data transfers of undefined byte lengths, byte count should be set to 0. When byte count is set to 0, it is the responsibility of host to explicitly send stop/abort command to terminate data transfer. (R/W)

Register 9.10: INTMASK_REG (0x0024)

(reserved)	SDIO_INT_MASK	INT_MASK	
31 18	17 16	15 0	
0x00000	0x00000	0x00000	Reset

SDIO_INT_MASK SDIO interrupt mask, one bit for each card. Bit[17:16] correspond to card[15:0] respectively. When masked, SDIO interrupt detection for that card is disabled. 0 masks an interrupt, and 1 enables an interrupt. In MMC-Ver3.3-only mode, these bits are always 0. (R/W)

INT_MASK These bits used to mask unwanted interrupts. A value of 0 masks interrupt, and a value of 1 enables the interrupt. (R/W)
Bit 15 (EBE): End-bit error, read/write (no CRC)
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation-by-host timeout/Volt_switch_int
Bit 9 (DRTO): Data read timeout
Bit 8 (RTO): Response timeout
Bit 7 (DCRC): Data CRC error
Bit 6 (RCRC): Response CRC error
Bit 5 (RXDR): Receive FIFO data request
Bit 4 (TXDR): Transmit FIFO data request
Bit 3 (DTO): Data transfer over
Bit 2 (CD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect

Register 9.11: CMDARG_REG (0x0028)

31 0	
0x000000000	Reset

CMDARG_REG Value indicates command argument to be passed to the card. (R/W)

Register 9.12: CMD_REG (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

START_CMD Start command. Once command is served by the CIU, this bit is automatically cleared. When this bit is set, host should not attempt to write to any command registers. If a write is attempted, hardware lock error is set in raw interrupt register. Once command is sent and a response is received from SD_MMC_CEATA cards, Command Done bit is set in the raw interrupt Register. (R/W)

USE_HOLE Use Hold Register. (R/W) 0: CMD and DATA sent to card bypassing HOLD Register; 1: CMD and DATA sent to card through the HOLD Register.

CCS_EXPECTED Expected Command Completion Signal (CCS) configuration. (R/W)

0: Interrupts are not enabled in CE-ATA device (nIEN = 1 in ATA control register), or command does not expect CCS from device.

1: Interrupts are enabled in CE-ATA device (nIEN = 0), and RW_BLK command expects command completion signal from CE-ATA device.

If the command expects Command Completion Signal (CCS) from the CE-ATA device, the software should set this control bit. SD/MMC sets Data Transfer Over (DTO) bit in RINTSTS register and generates interrupt to host if Data Transfer Over interrupt is not masked.

READ_CEATA_DEVICE Read access flag. (R/W)

0: Host is not performing read access (RW_REG or RW_BLK)towards CE-ATA device

1: Host is performing read access (RW_REG or RW_BLK) towards CE-ATA device.

Software should set this bit to indicate that CE-ATA device is being accessed for read transfer. This bit is used to disable read data timeout indication while performing CE-ATA read transfers. Maximum value of I/O transmission delay can be no less than 10 seconds. SD/MMC should not indicate read data timeout while waiting for data from CE-ATA device. (R/W)

Register 9.13: CMD_REG (continued) (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

UPDATE_CLOCK_REGISTERS_ONLY (R/W)

0: Normal command sequence.

1: Do not send commands, just update clock register value into card clock domain

Following register values are transferred into card clock domain: CLKDIV, CLRSRC, and CLKENA. Changes card clocks (change frequency, truncate off or on, and set low-frequency mode). This is provided in order to change clock frequency or stop clock without having to send command to cards.

During normal command sequence, when update_clock_registers_only = 0, following control registers are transferred from BIU to CIU: CMD, CMDARG, TMOUT, CTYPE, BLKSIZ, and BYTCNT. CIU uses new register values for new command sequence to card(s). When bit is set, there are no Command Done interrupts because no command is sent to SD_MMC_CEATA cards.

CARD_NUMBER Card number in use. Represents physical slot number of card being accessed. In MMC-Ver3.3-only mode, up to two cards are supported. In SD-only mode, up to two cards are supported. (R/W)

SEND_INITIALIZATION (R/W)

0: Do not send initialization sequence (80 clocks of 1) before sending this command.

1: Send initialization sequence before sending this command.

After power on, 80 clocks must be sent to card for initialization before sending any commands to card. Bit should be set while sending first command to card so that controller will initialize clocks before sending command to card.

STOP_ABORT_CMD (R/W)

0: Neither stop nor abort command can stop current data transfer. If abort is sent to function-number currently selected or not in data-transfer mode, then bit should be set to 0.

1: Stop or abort command intended to stop current data transfer in progress. When open-ended or predefined data transfer is in progress, and host issues stop or abort command to stop data transfer, bit should be set so that command/data state-machines of CIU can return correctly to idle state.

Register 9.14: CMD_REG (continued) (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

WAIT_PRVDATA_COMPLETE (R/W)

0: Send command at once, even if previous data transfer has not completed;

1: Wait for previous data transfer to complete before sending Command.

The wait_prvdata_complete = 0 option is typically used to query status of card during data transfer or to stop current data transfer. card_number should be same as in previous command.

SEND_AUTO_STOP (R/W)

0: No stop command is sent at the end of data transfer;

1: Send stop command at the end of data transfer.

TRANSFER_MODE (R/W)

0: Block data transfer command;

1: Stream data transfer command. Don't care if no data expected.

READ/WRITE (R/W)

0: Read from card;

1: Write to card.

Don't care if no data is expected from card.

DATA_EXPECTED (R/W)

0: No data transfer expected.

1: Data transfer expected.

CHECK_RESPONSE_CRC (R/W)

0: Do not check;

1: Check response CRC.

Some of command responses do not return valid CRC bits. Software should disable CRC checks for those commands in order to disable CRC checking by controller.

RESPONSE_LENGTH (R/W)

0: Short response expected from card;

1: Long response expected from card.

RESPONSE_EXPECT (R/W)

0: No response expected from card;

1: Response expected from card.

CMD_INDEX Command index. (R/W)

Register 9.15: RESP0_REG (0x0030)

31	0	
0x000000000		Reset

RESP0_REG Bit[31:0] of response. (RO)

Register 9.16: RESP1_REG (0x0034)

31	0	
0x000000000		Reset

RESP1_REG Bit[63:32] of long response. (RO)

Register 9.17: RESP2_REG (0x0038)

31	0	
0x000000000		Reset

RESP2_REG Bit[95:64] of long response. (RO)

Register 9.18: RESP3_REG (0x003C)

31	0	
0x000000000		Reset

RESP3_REG Bit[127:96] of long response. (RO)

Register 9.19: MINTSTS_REG (0x0040)

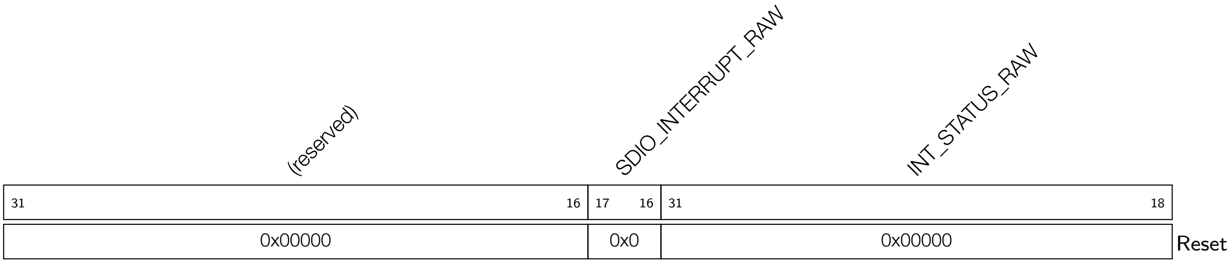
(reserved)	SDIO_INTERRUPT_MSK	INT_STATUS_MSK	
31 18	17 16	15 0	
0	0x0	0x00000	Reset

SDIO_INTERRUPT_MSK Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. SDIO interrupt for card is enabled only if corresponding sdio_int_mask bit is set in Interrupt mask register (Setting mask bit enables interrupt). (RO)

INT_STATUS_MSK Interrupt enabled only if corresponding bit in interrupt mask register is set. (RO)

Bit 15 (EBE): End-bit error, read/write (no CRC)
Bit 14 (ACD): Auto command done
Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt
Bit 12 (HLE): Hardware locked write error
Bit 11 (FRUN): FIFO underrun/overrun error
Bit 10 (HTO): Data starvation by host timeout (HTO)
Bit 9 (DTRO): Data read timeout
Bit 8 (RTO): Response timeout
Bit 7 (DCRC): Data CRC error
Bit 6 (RCRC): Response CRC error
Bit 5 (RXDR): Receive FIFO data request
Bit 4 (TXDR): Transmit FIFO data request
Bit 3 (DTO): Data transfer over
Bit 2 (CD): Command done
Bit 1 (RE): Response error
Bit 0 (CD): Card detect

Register 9.20: RINTSTS_REG (0x0044)



SDIO_INTERRUPT_RAW Interrupt from SDIO card, one bit for each card. Bit[17:16] correspond to card1 and card0, respectively. Setting a bit clears the corresponding interrupt bit and writing 0 has no effect. (R/W)

0: No SDIO interrupt from card;

1: SDIO interrupt from card.

In MMC-Ver3.3-only mode, these bits are always 0. Bits are logged regardless of interrupt-mask status. (R/W)

INT_STATUS_RAW Setting a bit clears the corresponding interrupt and writing 0 has no effect. Bits are logged regardless of interrupt mask status. (R/W)

Bit 15 (EBE): End-bit error, read/write (no CRC)

Bit 14 (ACD): Auto command done

Bit 13 (SBE/BCI): Start Bit Error/Busy Clear Interrupt

Bit 12 (HLE): Hardware locked write error

Bit 11 (FRUN): FIFO underrun/overrun error

Bit 10 (HTO): Data starvation by host timeout (HTO)

Bit 9 (DTRO): Data read timeout

Bit 8 (RTO): Response timeout

Bit 7 (DCRC): Data CRC error

Bit 6 (RCRC): Response CRC error

Bit 5 (RXDR): Receive FIFO data request

Bit 4 (TXDR): Transmit FIFO data request

Bit 3 (DTO): Data transfer over

Bit 2 (CD): Command done

Bit 1 (RE): Response error

Bit 0 (CD): Card detect

Register 9.21: STATUS_REG (0x0048)

(reserved)	(reserved)	FIFO_COUNT	RESPONSE_INDEX	DATA_STATE_MC_BUSY	DATA_BUSY	DATA_3_STATUS	COMMAND_FSM_STATES	FIFO_FULL	FIFO_EMPTY	FIFO_TX_WATERMARK	FIFO_RX_WATERMARK	
31	30	29 17	16 11	10	9	8	7 4	3	2	1	0	
0	0	0x000	0x00	1	1	1	0x01	0	1	1	0	Reset

FIFO_COUNT FIFO count, number of filled locations in FIFO. (RO)

RESPONSE_INDEX Index of previous response, including any auto-stop sent by core. (RO)

DATA_STATE_MC_BUSY Data transmit or receive state-machine is busy. (RO)

DATA_BUSY Inverted version of raw selected card_data[0]. (RO)
0: Card data not busy;
1: Card data busy.

DATA_3_STATUS Raw selected card_data[3], checks whether card is present. (RO)
0: card not present;
1: card present.

COMMAND_FSM_STATES Command FSM states. (RO)
0: Idle
1: Send init sequence
2: Send cmd start bit
3: Send cmd tx bit
4: Send cmd index + arg
5: Send cmd crc7
6: Send cmd end bit
7: Receive resp start bit
8: Receive resp IRQ response
9: Receive resp tx bit
10: Receive resp cmd idx
11: Receive resp data
12: Receive resp crc7
13: Receive resp end bit
14: Cmd path wait NCC
15: Wait, cmd-to-response turnaround

FIFO_FULL FIFO is full status. (RO)

FIFO_EMPTY FIFO is empty status. (RO)

FIFO_TX_WATERMARK FIFO reached Transmit watermark level, not qualified with data transfer. (RO)

FIFO_RX_WATERMARK FIFO reached Receive watermark level, not qualified with data transfer. (RO)

Register 9.22: FIFOTH_REG (0x004C)

(reserved)	DMA_MULTIPLE_TRANSACTION_SIZE	(reserved)	RX_WMARK	(reserved)	TX_WMARK	
31	30 28	27	26 16	15 12	11 0	
0	0x0	0	x x x x x x x x x x x	0 0 0 0	0x0000	Reset

DMA_MULTIPLE_TRANSACTION_SIZE Burst size of multiple transaction, should be programmed same as DMA controller multiple-transaction-size SRC/DEST_MSIZE. 000: 1-byte transfer; 001: 4-byte transfer; 010: 8-byte transfer; 011: 16-byte transfer; 100: 32-byte transfer; 101: 64-byte transfer; 110: 128-byte transfer; 111: 256-byte transfer. (R/W)

RX_WMARK FIFO threshold watermark level when receiving data to card.When FIFO data count reaches greater than this number (FIFO_RX_WATERMARK), DMA/FIFO request is raised. During end of packet, request is generated regardless of threshold programming in order to complete any remaining data.In non-DMA mode, when receiver FIFO threshold (RXDR) interrupt is enabled, then interrupt is generated instead of DMA request.During end of packet, interrupt is not generated if threshold programming is larger than any remaining data. It is responsibility of host to read remaining bytes on seeing Data Transfer Done interrupt.In DMA mode, at end of packet, even if remaining bytes are less than threshold, DMA request does single transfers to flush out any remaining bytes before Data Transfer Done interrupt is set. (R/W)

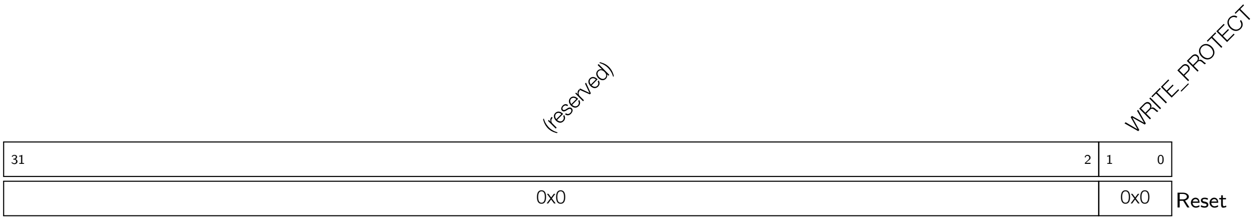
TX_WMARK FIFO threshold watermark level when transmitting data to card. When FIFO data count is less than or equal to this number (FIFO_TX_WATERMARK), DMA/FIFO request is raised. If Interrupt is enabled, then interrupt occurs. During end of packet, request or interrupt is generated, regardless of threshold programming.In non-DMA mode, when transmit FIFO threshold (TXDR) interrupt is enabled, then interrupt is generated instead of DMA request. During end of packet, on last interrupt, host is responsible for filling FIFO with only required remaining bytes (not before FIFO is full or after CIU completes data transfers, because FIFO may not be empty). In DMA mode, at end of packet, if last transfer is less than burst size, DMA controller does single cycles until required bytes are transferred. (R/W)

Register 9.23: CDETECT_REG (0x0050)

(reserved)	CARD_DETECT_N	
31 2	1 0	
0x0	0x0	Reset

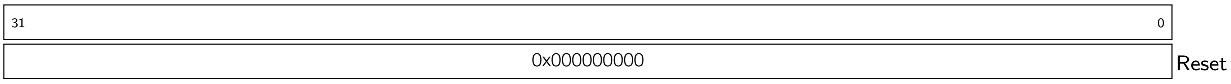
CARD_DETECT_N Value on card_detect_n input ports (1 bit per card), read-only bits.0 represents presence of card. Only NUM_CARDS number of bits are implemented. (RO)

Register 9.24: WRTPRT_REG (0x0054)



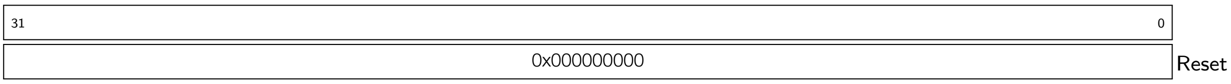
WRITE_PROTECT Value on card_write_prt input ports (1 bit per card).1 represents write protection. Only NUM_CARDS number of bits are implemented. (RO)

Register 9.25: TCBCNT_REG (0x005C)



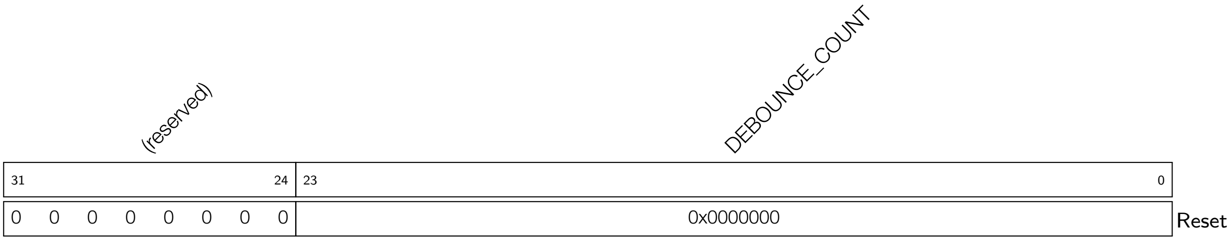
TCBCNT_REG Number of bytes transferred by CIU unit to card. (RO)

Register 9.26: TBBCNT_REG (0x0060)



TBBCNT_REG Number of bytes transferred between Host/DMA memory and BIU FIFO. (RO)

Register 9.27: DEBNCE_REG (0x0064)



DEBOUNCE_COUNT Number of host clocks (clk) used by debounce filter logic. The typical debounce time is 5 ~ 25 ms to prevent the card instability when the card is inserted or removed. (R/W)

Register 9.28: USRID_REG (0x0068)

31 0	
0x000000000	Reset

USRID_REG User identification register, value set by user. Default reset value can be picked by user while configuring core before synthesis. Can also be used as a scratchpad register by user. (R/W)

Register 9.29: RST_N_REG (0x0078)

(reserved)	RST_CARD_RESET	
31 2	1 0	
0	0x1	Reset

RST_CARD_RESET Hardware reset.1: Active mode; 0: Reset. These bits cause the cards to enter pre-idle state, which requires them to be re-initialized. CARD_RESET[0] should be set to 1'b0 to reset card0, CARD_RESET[1] should be set to 1'b0 to reset card1.The number of bits implemented is restricted to NUM_CARDS. (R/W)

Register 9.30: BMOD_REG (0x0080)

(reserved)	BMOD_PBL	BMOD_DE	(reserved)	BMOD_FB	BMOD_SWR	
31 11	10 8	7	6 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0	0x00	0	0	Reset

BMOD_PBL Programmable Burst Length. These bits indicate the maximum number of beats to be performed in one IDMAC transaction. The IDMAC will always attempt to burst as specified in PBL each time it starts a burst transfer on the host bus. The permissible values are 1, 4, 8, 16, 32, 64, 128 and 256. This value is the mirror of MSIZE of FIFOTH register. In order to change this value, write the required value to FIFOTH register. This is an encode value as follows:
000: 1-byte transfer; 001: 4-byte transfer; 010: 8-byte transfer; 011: 16-byte transfer; 100: 32-byte transfer; 101: 64-byte transfer; 110: 128-byte transfer; 111: 256-byte transfer.
PBL is a read-only value and is applicable only for data access, it does not apply to descriptor access. (R/W)

BMOD_DE IDMAC Enable. When set, the IDMAC is enabled. (R/W)

BMOD_FB Fixed Burst. Controls whether the AHB Master interface performs fixed burst transfers or not. When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB will use SINGLE and INCR burst transfer operations. (R/W)

BMOD_SWR Software Reset. When set, the DMA Controller resets all its internal registers. It is automatically cleared after one clock cycle. (R/W)

Register 9.31: PLDMND_REG (0x0080)

31	0	
0x000000000		Reset

PLDMND_REG Poll Demand. If the OWN bit of a descriptor is not set, the FSM goes to the Suspend state. The host needs to write any value into this register for the IDMAC FSM to resume normal descriptor fetch operation. This is a write only register, PD bit is write-only. (WO)

Register 9.32: DBADDR_REG (0x0088)

31	0	
0x000000000		Reset

DBADDR_REG Start of Descriptor List. Contains the base address of the First Descriptor. The LSB bits [1:0] are ignored and taken as all-zero by the IDMAC internally. Hence these LSB bits may be treated as read-only. (R/W)

Register 9.33: IDSTS_REG (0x008C)

(reserved)	IDSTS_FSM	IDSTS_FBE_CODE	IDSTS_AIS	IDSTS_NIS	(reserved)	IDSTS_CES	IDSTS_DU	(reserved)	IDSTS_FBE	IDSTS_RI	IDSTS_TI	
31 17	16 13	12 10	9	8	7 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	0x0	0	0	0 0	0	0	0	0	0	0	Reset

IDSTS_FSM DMAC FSM present state: (RO)

0: DMA_IDLE; 1: DMA_SUSPEND; 2: DESC_RD; 3: DESC_CHK; 4: DMA_RD_REQ_WAIT
5: DMA_WR_REQ_WAIT; 6: DMA_RD; 7: DMA_WR; 8: DESC_CLOSE.

IDSTS_FBE_CODE Fatal Bus Error Code. Indicates the type of error that caused a Bus Error. Valid only when the Fatal Bus Error bit IDSTS[2] is set. This field does not generate an interrupt. (RO)

3b001: Host Abort received during transmission;
3b010: Host Abort received during reception;
Others: Reserved.

IDSTS_AIS Abnormal Interrupt Summary. Logical OR of the following: IDSTS[2] : Fatal Bus Interrupt, IDSTS[4] : DU bit Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes AIS to be set is cleared. Writing 1 clears this bit. (R/W)

IDSTS_NIS Normal Interrupt Summary. Logical OR of the following: IDSTS[0] : Transmit Interrupt, IDSTS[1] : Receive Interrupt. Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes NIS to be set is cleared. Writing 1 clears this bit. (R/W)

IDSTS_CES Card Error Summary. Indicates the status of the transaction to/from the card, also present in RINTSTS. Indicates the logical OR of the following bits: EBE : End Bit Error, RTO : Response Timeout/Boot Ack Timeout, RCRC : Response CRC, SBE : Start Bit Error, DRTO : Data Read Timeout/BDS timeout, DCRC : Data CRC for Receive, RE : Response Error.

Writing 1 clears this bit. The abort condition of the IDMAC depends on the setting of this CES bit. If the CES bit is enabled, then the IDMAC aborts on a response error. (R/W)

IDSTS_DU Descriptor Unavailable Interrupt. This bit is set when the descriptor is unavailable due to OWN bit = 0 (DES0[31] =0). Writing 1 clears this bit. (R/W)

IDSTS_FBE Fatal Bus Error Interrupt. Indicates that a Bus Error occurred (IDSTS[12:10]) . When this bit is set, the DMA disables all its bus accesses. Writing 1 clears this bit. (R/W)

IDSTS_RI Receive Interrupt. Indicates the completion of data reception for a descriptor. Writing 1 clears this bit. (R/W)

IDSTS_TI Transmit Interrupt. Indicates that data transmission is finished for a descriptor. Writing 1 clears this bit. (R/W)

Register 9.34: IDINTEN_REG (0x0090)

(reserved)	IDINTEN_AI	IDINTEN_NI	(reserved)	IDINTEN_CES	IDINTEN_DU	(reserved)	IDINTEN_FBE	IDINTEN_RI	IDINTEN_TI	
31 10	9	8	7 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0	0	0	0	0	0	0	Reset

IDINTEN_AI Abnormal Interrupt Summary Enable. (R/W)
When set, an abnormal interrupt is enabled. This bit enables the following bits:
IDINTEN[2]: Fatal Bus Error Interrupt;
IDINTEN[4]: DU Interrupt.

IDINTEN_NI Normal Interrupt Summary Enable. (R/W)
When set, a normal interrupt is enabled. When reset, a normal interrupt is disabled. This bit enables the following bits:
IDINTEN[0]: Transmit Interrupt;
IDINTEN[1]: Receive Interrupt.

IDINTEN_CES Card Error summary Interrupt Enable. When set, it enables the Card Interrupt summary. (R/W)

IDINTEN_DU Descriptor Unavailable Interrupt. When set along with Abnormal Interrupt Summary Enable, the DU interrupt is enabled. (R/W)

IDINTEN_FBE Fatal Bus Error Enable. When set with Abnormal Interrupt Summary Enable, the Fatal Bus Error Interrupt is enabled. When reset, Fatal Bus Error Enable Interrupt is disabled. (R/W)

IDINTEN_RI Receive Interrupt Enable. When set with Normal Interrupt Summary Enable, Receive Interrupt is enabled. When reset, Receive Interrupt is disabled. (R/W)

IDINTEN_TI Transmit Interrupt Enable. When set with Normal Interrupt Summary Enable, Transmit Interrupt is enabled. When reset, Transmit Interrupt is disabled. (R/W)

Register 9.35: DSCADDR_REG (0x0094)

31 0	
0x000000000	Reset

DSCADDR_REG Host Descriptor Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the start address of the current descriptor read by the IDMAC. (RO)

Register 9.36: BUFADDR_REG (0x0098)

31 0	
0x000000000	Reset

BUFADDR_REG Host Buffer Address Pointer, updated by IDMAC during operation and cleared on reset. This register points to the current Data Buffer Address being accessed by the IDMAC. (RO)

Register 9.37: CLK_EDGE_SEL (0x0800)

(reserved)	CCLKIN_EDGE_N	CCLKIN_EDGE_L	CCLKIN_EDGE_H	CCLKIN_EDGE_SLF_SEL	CCLKIN_EDGE_SAM_SEL	CCLKIN_EDGE_DRV_SEL	
31 21	20 17	16 13	12 9	8 6	5 3	2 0	
0x000	0x1	0x0	0x1	0x0	0x0	0x0	Reset

CCLKIN_EDGE_N This value should be equal to CCLKIN_EDGE_L. (R/W)

CCLKIN_EDGE_L The low level of the divider clock. The value should be larger than CCLKIN_EDGE_H. (R/W)

CCLKIN_EDGE_H The high level of the divider clock. The value should be smaller than CCLKIN_EDGE_L. (R/W)

CCLKIN_EDGE_SLF_SEL It is used to select the clock phase of the internal signal from phase90, phase180, or phase270. (R/W)

CCLKIN_EDGE_SAM_SEL It is used to select the clock phase of the input signal from phase90, phase180, or phase270. (R/W)

CCLKIN_EDGE_DRV_SEL It is used to select the clock phase of the output signal from phase90, phase180, or phase270. (R/W)

10. Ethernet MAC

10.1 Overview

Features of Ethernet

By using the external Ethernet PHY (physical layer), ESP32 can send and receive data via Ethernet MAC (Media Access Controller) according to the IEEE 802.3 standard, as Figure 40 shows. Ethernet is currently the most commonly used network protocol that controls how data is transmitted over local- and wide-area networks, abbreviated as LAN and WAN, respectively.

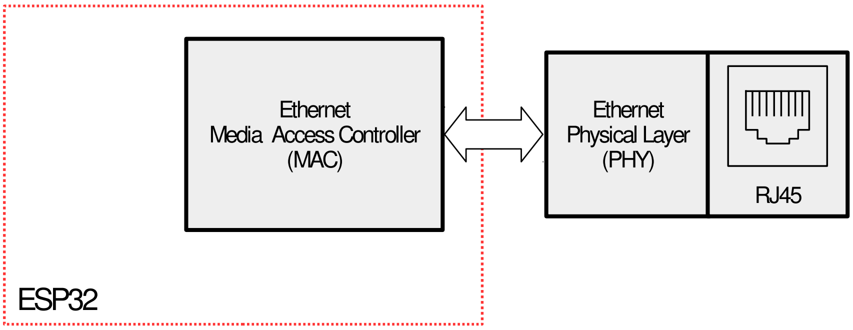


Figure 40: Ethernet MAC Functionality Overview

ESP32 MAC Ethernet complies with the following criteria:

- IEEE 802.3-2002 for Ethernet MAC
- IEEE 1588-2008 standard for specifying the accuracy of networked clock synchronization
- Two industry-standard interfaces conforming with IEEE 802.3-2002: Media-Independent Interface (MII) and Reduced Media-Independent Interface (RMII).

Features of MAC Layer

- Support for a data transmission rate of 10 Mbit/s or 100 Mbit/s through an external PHY interface
- Communication with an external Fast Ethernet PHY through IEEE 802.3-compliant MII and RMII interfaces
- Support for:
  - Carrier Sense Multiple Access / Collision Detection (CSMA/CD) protocol in half-duplex mode
  - IEEE 802.3x flow control in full-duplex mode
  - operations in full-duplex mode, forwarding the received pause-control frame to the user application
  - backpressure flow control in half-duplex mode
  - If the flow control input signal disappears during a full-duplex operation, a pause frame with zero pause time value is automatically transmitted.
- The Preamble and the Start Frame Delimiter (SFD) are inserted in the Transmit path, and deleted in the Receive path.
- Cyclic Redundancy Check (CRC) and Pad can be controlled on a per-frame basis.
- The Pad is generated automatically, if data is below the minimum frame length.

- Programmable frame length supporting jumbo frames of up to 16 KB
- Programmable Inter-frame Gap (IFG) (40-96 bit times in steps of 8)
- Support for a variety of flexible address filtering modes:
  - Up to eight 48-bit perfect address filters to mask each byte
  - Up to eight 48-bit SA address comparison checks to mask each byte
  - All multicast address frames can be transmitted
  - All frames in mixed mode can be transmitted without being filtered for network monitoring
  - A status report is attached each time all incoming packets are transmitted and filtered.
- Returning a 32-bit status for transmission and reception of packets respectively
- Separate transmission, reception, and control interfaces for the application
- Use of the Management Data Input/Output (MDIO) interface to configure and manage PHY devices
- Support for the offloading of received IPv4 and TCP packets encapsulated by an Ethernet frame in the reception function
- Support for checking IPv4 header checksums, as well as TCP, UDP, or ICMP (Internet Control Message Protocol) checksums encapsulated in IPv4/IPv6 packets in the enhanced reception function
- Support for Ethernet frame timestamps. (For details please refer to IEEE 1588-2008.) Each frame has a 64-bit timestamp when transmitted or received.
- Two sets of FIFOs: one 2 KB Tx FIFO with programmable threshold and one 2 KB Rx FIFO with configurable threshold (64 bytes by default)
- When Rx FIFO stores multiple frames, the Receive Status Vector is inserted into the Rx FIFO after transmitting an EOF (end of frame), so that the Rx FIFO does not need to store the Receive Status of these frames.
- In store-and-forward mode, all error frames can be filtered during reception, but not forwarded to the application.
- Under-sized good frames can be forwarded.
- Support for data statistics by generating pulses for lost or corrupted frames in the Rx FIFO due to an overflow
- Support for store-and-forward mechanism when transmitting data to the MAC core
- Automatic re-transmission of collided frames during transmission (subject to certain conditions, see section 10.2.1.2)
- Discarding frames in cases of late collisions, excessive collisions, excessive deferrals, and under-run conditions
- The Tx FIFO is flushed by software control.
- Calculating the IPv4 header checksum, as well as the TCP, UDP, or ICMP checksum, and then inserting them into frames transmitted in store-and-forward mode.

Ethernet Block Diagram

Figure 41 shows the block diagram of the Ethernet.

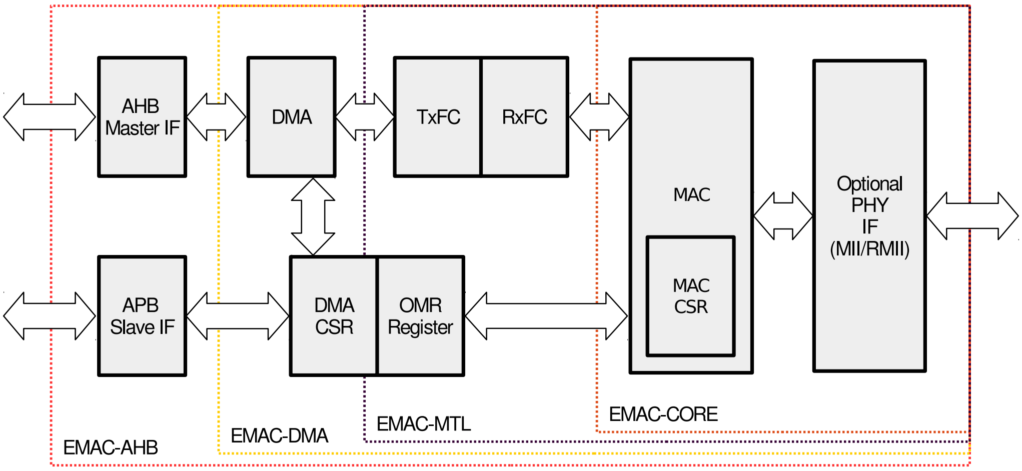


Figure 41: Ethernet Block Diagram

Ethernet MAC consists of the MAC-layer configuration register module and three layers: EMAC_CORE (MAC Core Layer), EMAC_MTL (MAC Transition Layer), and EMAC_DMA (Direct Memory Access). Each of these three layers has two directions: Tx and Rx. They are connected to the system through the Advanced High-Performance Bus (AHB) and the Advanced Peripheral Bus (APB) on the chip. Off the chip, they communicate with the external PHY through the MII and RMII interfaces to materialize an Ethernet connection.

10.2 EMAC_CORE

The MAC supports many interfaces with the PHY chip. The PHY interface can be selected only once after reset. The MAC communicates with the application side (DMA side), using the MAC Transmit Interface (MTI), MAC Receive Interface (MRI) and the MAC Control Interface (MCI).

10.2.1 Transmit Operation

A transmit operation is initiated when the MTL Application pushes in data at the time a response signal is asserted. When the SOF (start of frame) signal is detected, the MAC accepts the data and begins transmitting to the RMII or MII. The time required to transmit the frame data to the RMII or MII, after the application initiates transmission, varies, depending on delay factors like IFG delay, time to transmit Preamble or SFD (Start Frame Delimiter), and any back-off delays in half-duplex mode. Until then, the MAC does not accept the data received from MTL by de-asserting the ready signal.

After the EOF (end of frame) is transmitted to the MAC, the MAC completes the normal transmission and yields the Transmit Status to the MTL. If a normal collision (in half-duplex mode) occurs during transmission, the MAC makes valid the Transmit Status in the MTL. It then accepts and drops all further data until the next SOF is received. The MTL block should retransmit the same frame from SOF upon observing a retry request (in the Status) from the MAC.

The MAC issues an underflow status if the MTL is not able to provide the data continuously during transmission. During the normal transmission of a frame from MTL, if the MAC receives an SOF without getting an EOF for the previous frame, it ignores the SOF and considers the new frame as a continuation of the previous one.

10.2.1.1 Transmit Flow Control

In full-duplex mode, when the Transmit Flow Control Enable bit (TFE bit in the Flow Control Register) is set to 1, the MAC will generate and send a pause frame, as needed. The pause frame is added and transmitted together with the calculated CRC. The generation of pause frames can be initiated in two ways.

When the application sets the Flow Control Busy bit (FCB bit in the Flow Control Register) to 1, or when the Rx FIFO is full, a pause frame is transmitted.

- If an application has requested flow control by setting the FCB bit in the Flow Control Register to 1, the MAC will generate and send a single pause frame. The pause time value in the generated frame is the pause time value programmed in the Flow Control Register. To extend or end the pause time before the time specified in the previously transmitted pause frame, the application program must configure the pause time value in the Flow Control Register to the appropriate value and, then, request another pause frame transmission.
- If the application has requested flow control when the Rx FIFO is full, the MAC will generate and transmit a pause frame. The value of the pause time of the generated frame is the pause time value programmed in the Flow Control Register. If the Rx FIFO remains full during the configurable interval, which is determined by the Pause Low Threshold bit (PLT) in the Flow Control Register before the pause time expires, a second pause frame will be transmitted. As long as the Rx FIFO remains full, the process repeats itself. If the FIFO is no longer full before the sample time, the MAC will send a pause frame with zero pause time, indicating to the remote end that the Rx buffer is ready to receive the new data frame.

10.2.1.2 Retransmission During a Collision

In half-duplex mode, a collision may occur on the MAC line interface when frames are transmitted to the MAC. The MAC may even give a status to indicate a retry before the end of the frame is received. The retransmission is then enabled and the frame is popped out from the FIFO. When more than 96 bytes are transmitted to the MAC core, the FIFO controller frees the space in the FIFO, allowing the DMA to push more data into FIFO. This means that data cannot be retransmitted after the threshold is exceeded or when the MAC core indicates that a late collision has occurred.

The MAC transmitter may abort the transmission of a frame because of collision, Tx FIFO underflow, loss of carrier, jabber timeout, no carrier, excessive deferral, and late collision. When frame transmission is aborted because of collision, the MAC requests retransmission of the frame.

10.2.2 Receive Operation

A receive operation is initiated when the MAC detects an SFD on the RMII or MII. The MAC strips the Preamble and SFD before processing the frame. The header fields are checked for the filtering and the FCS (Frame Check Sequence) field used to verify the CRC for the frame. The received frame is stored in a shallow buffer until the address filtering is performed. The frame is dropped in the MAC if it fails the address filtering.

The frame received by the MAC will be pushed into the Rx FIFO. Once the FIFO status exceeds the Receive Threshold, configured by the Receive Threshold Control (RTC) bit in the Operation Mode register, the DMA can initiate a preconfigured burst transmission to the AHB interface.

In the default pass-through mode, when the FIFO receives a complete packet or 64 bytes configured by the RTC bit in the Operation Mode Register, the data pops up and its availability is notified to the DMA. After the DMA initiates the transmission to the AHB interface, the data transmission continues from the FIFO until the complete packet is transmitted. Upon completing transmitting the EOF, the status word will pop up and be transmitted to the DMA controller.

In the Rx FIFO Store-and-Forward mode (configured through the RSF or Receive Store and Forward bit in the Operation Mode Register), only the valid frames are read and forwarded to the application. In the passthrough mode, error frames are not discarded because the error status is received at the end of the frame. The start of frame will have been read from the FIFO at that point.

10.2.2.1 Reception Protocol

After the receive module receives the packets, the Preamble and SFD of the received frames are removed. When the SFD is detected, the MAC starts sending Ethernet frame data to the Rx FIFO, starting at the first byte (destination address) following the SFD. This timestamp is passed on to the application, unless the MAC filters out and drops the frame.

If the received frame length/type is less than 0x600 and the automatic CRC/Pad removal option is programmed for the MAC, the MAC will send frame data to the Rx FIFO (the amount of data does not exceed the number specified in the length/type field). Then MAC begins discarding the remaining section, including the FCS field. If the frame length/type is greater than, or equal to, 0x600, the MAC will send all received Ethernet frame data to the Rx FIFO, regardless of the programmed value of the automatic CRC removal option. By default, the MAC watchdog timer is enabled, meaning that frames, including DA, SA, LT, data, pad and FCS, which exceed 2048 bytes, are cut off. This function can be disabled by programming the Watchdog Disable (WD) bit in the MAC Configuration Register. However, even if the watchdog timer is disabled, frames longer than 16 KB will be cut off and the watchdog timeout status will be given.

10.2.2.2 Receive Frame Controller

If the RA (Receive All) bit in the MAC Frame Filter Register is reset, the MAC will filter frames based on the destination and source addresses. If the application decides not to receive any bad frames, such as runt frames and CRC error frames, another level of filtering is needed. When a frame fails the filtering, the frame is discarded and is not transmitted to the application. When the filter parameters are changed dynamically, if a frame fails the DA and SA filterings, the remaining part of the frame is discarded and the Receive Status word is updated immediately and, therefore, the zero frame length bit, CRC error bit, and runt frame error bit are set to 1. This indicates that the frame has failed the filtering.

10.2.2.3 Receive Flow Control

The MAC will detect the received pause frame and pause transmission of frames for a specified delay within the received pause frame (in full-duplex mode only). The Pause Frame Detect Function can be enabled or disabled by the RFCE (Receive Flow Control Enable) bit in the Flow Control Register. When receive flow control is enabled, it starts monitoring whether the destination address of the received frame matches the multicast address of the control frame (0x0180 C200 0001). If a match is detected (i.e. the destination address of the received frame matches the destination address of the reserved control frame), the MAC will determine whether to transmit the received control frame to the application, according to the PCF (Pass Control Frames) bit in the Frame Filter Register.

The MAC will also decode the type, the opcode, and the pause timer field of the Receive Control Frame. If the value of the status byte counter is 64 bits and there are no CRC errors, the MAC transmitter will halt the transmission of any data frame. The duration of the pause is the decoded pause time value multiplied by the interval (which is 64 bytes for both 10 Mbit/s and 100 Mb/s modes). At the same time, if another pause frame of zero pause time is detected, the MAC will reset the pause time to manage the new pause request.

If the type field (0x8808), the opcode (0x00001), and the byte length (64 bytes) of the received control frame are not 0x8808, 0x00001, and 64 bytes, respectively, or if there is a CRC error, the MAC will not generate a pause.

If a pause frame has a multicast destination address, the MAC filters the frame, according to the address matching.

For pause frames with a unicast destination address, the MAC checks whether the DA matches the content of the EMACADDR0 Register, and whether the Unicast Pause Frame Detect (UPFD) bit in the Flow Control Register is set to 1. The Pass Control Frames (PCF) bits in the Frame Filter Register [7:6] control the filtering of frames and addresses.

10.2.2.4 Reception of Multiple Frames

Since the status is available immediately after the data is received. Frames can be stored there, as long as the FIFO is not full.

10.2.2.5 Error Handling

If the Rx FIFO is full before receiving the EOF data from the MAC, an overflow will be generated and the entire frame will be discarded. In fact, status bit RDES0[11] will indicate that this frame is partial due to an overflow, and that it should be discarded.

If the function that corresponds to the Flush Transmit FIFO (FTF) bit and the Forward Undersized Good Frames (FUGF) bit in the Operation Mode Register is enabled, the Rx FIFO can filter error frames and runt frames. If the receive FIFO is configured to operate in store-and-forward mode, all error frames will be filtered and discarded.

In passthrough mode, if a frame's status and length are available when reading a SOF from the Rx FIFO, the entire error frame can be discarded. DMA can clear the error frame being read from the FIFO by enabling the Receive Frame Clear bit. The data transmission to the application (DMA) will then stop, and the remaining frames will be read internally and discarded. If FIFO is available, the transmission of the next frame will be initiated.

10.2.2.6 Receive Status Word

After receiving the Ethernet frames, the MAC outputs the receive status to the application. The detailed description of the receive status is the same as that which is configured by bit [31:0] in RDES0.

10.3 MAC Interrupt Controller

The MAC core can generate interrupts due to various events.

The interrupt register bits only indicate various interrupt events. To clear the interrupts, the corresponding status register and other registers must be read. An Interrupt Status Register describes the events that prompt the MAC core to generate interrupts. Each interrupt event can be prevented by setting the corresponding mask bit in the Interrupt Mask Register to 1. For example, if bit3 of the interrupt register is set high, it indicates that a magic packet or Wake-on-LAN frame has been received in Power-down mode. The PMT Control and Status register must be read to clear this interrupt event.

10.4 MAC Address Filtering

Address filtering will check the destination and source addresses of all received frames and report the address filtering status accordingly. For example, filtered frames can be identified either as multicast or broadcast.The address check, then, is based on the parameters selected by the application (Frame Filter Registers).

Physical (MAC) addresses are used for address checking during address filtering.

10.4.1 Unicast Destination Address Filtering

The MAC supports up to 8 MAC addresses for perfect filtering of unicast addresses. If a perfect filtering is selected (by resetting bit[1] in the Frame Filter Register), the MAC compares all 48 bits of the received unicast address with the programmed MAC address to determine if there is a match. By default, EMACADDR0 is always enabled, and the other addresses (EMACADDR0 ~ EMACADDR7) are selected by a separate enable bit. When the individual bytes of the other addresses (EMACADDR0 ~ EMACADDR7) are compared with the DA bytes received, the latter can be masked by setting the corresponding Mask Byte Control bit in the register to 1. This facilitates the DA group address filtering.

10.4.2 Multicast Destination Address Filtering

The MAC can be programmed to pass all multicast frames by setting the Pass All Multicast (PAM) bit in the Frame Filter Register to 1. If the PAM bit is reset, the MAC will filter multicast addresses, according to Bit[2] in the Frame Filter Register.

In perfect filtering mode, the multicast address is compared with the programmed MAC Destination Address Registers (EMACADDR0 ~ EMACADDR7). Group address filtering is also supported.

10.4.3 Broadcast Address Filtering

The MAC does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames, which can happen by setting the Disable Broadcast Frames (DBF) bit in the Frame Filter Register to 1, all broadcast frames will be discarded.

10.4.4 Unicast Source Address Filtering

The MAC may also perform a perfect filtering based on the source address field of the received frame. By default, the Address Filtering Module (AFM) compares the Source Address (SA) field with the values programmed in the SA register. By setting Bit[30] in the SA register to 1, the MAC Address Register (EMACADDR0 - EMACADDR7) can be configured to contain SA, instead of Destination Address (DA), for filtering. Group filtering with SA is also supported. If the Source Address Filter (SAF) enable bit in the Frame Filter Register is set to 1, the MAC discards frames that do not pass the SA filtering. Otherwise, the result of SA filtering is given as a status bit in the Receive Status word (Please refer to Table 46).

When the SAF enable bit is set to 1, the result of the SA filtering and DA filtering is AND'ed to determine whether or not to forward the frame. Any frame that fails to pass will be discarded. Frames need to pass both filterings in order to be forwarded to the application.

10.4.5 Inverse Filtering Operation

For both destination address (DA) and source address (SA) filtering, you can invert the results matched through the filtering at the final output. The inverse filtering of DA and SA are controlled by the DAIF and SAIF bits, respectively, in the Frame Filter Register. The DAIF bit applies to both unicast and multicast DA frames. When DAIF is set to 1, the result of unicast or multicast destination address filtering will be inverted. Similarly, when the SAIF bit is set to 1, the result of unicast SA filtering is reversed.

The following two tables summarize the destination address and source address filtering, based on the type of the frames received.

Table 38: Destination Address Filtering

Frame Type	PM	PF	DAIF	PAM	DB	DA Filter Result
Broadcast	1	X	X	X	X	Pass
	0	X	X	X	0	Pass
	0	X	X	X	1	Fail
Unicast	1	X	X	X	X	All frames pass.
	0	X	0	X	X	Pass when results of perfect/group filtering match.
	0	X	1	X	X	Fail when results of perfect/group filtering match.
	0	1	0	X	X	Pass when results of perfect/group filtering match.
	0	1	1	X	X	Fail when results of perfect/group filtering match.
Multicast	1	X	X	X	X	All frames pass.
	X	X	X	1	X	All frames pass.
	0	X	0	0	X	Pass when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x.
	0	1	0	0	X	Pass when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x.
	0	X	1	0	X	Fail when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x.
	0	1	1	0	X	Fail when results of perfect/group filtering match and pause control frame is discarded, if PCF = 0x.

The filtering parameters in the MAC Frame Filter Register described in Table 38 are as follows.

Parameter name:

PM:	Pass All Multicast
PF:	Perfect Filter
DAIF:	Destination Address Inverse Filtering
PAM:	Pass All Multicast
DB:	Disable Broadcast Frames

Parameter setting:

1:	Set
0:	Cleared

Table 39: Source Address Filtering

Frame Type	PM	SAIF	SAF	Source Address Filter Operation
Unicast	1	X	X	Pass all frames
	0	0	0	Pass when results of perfect/group filtering match. Frames not passed are not discarded.
	0	1	0	Fail when results of perfect/group filtering match. Frames not passed are not discarded.
	0	0	1	Pass when results of perfect/group filtering match. Frames not passed are discarded.
	0	1	1	Fail when results of perfect/group filtering match. Frames not passed are discarded.

The filtering parameters in the MAC Frame Filter Register described in Table 39 are as follows.

Parameter name:		**Parameter setting:**	
PM:	Pass All Multicast	1:	Set
SAF:	Source Address Filtering	0:	Cleared
SAIF:	Source Address Inverse Filtering	X:	Don't care

10.4.6 Good Transmitted Frames and Received Frames

A frame successfully transmitted is considered a ”good frame”. In other words, a transmitted frame is considered to be good, if the frame transmission is not aborted due to the following errors:

- Jabber timeout
- No carrier or loss of carrier
- Late collision
- Frame underflow
- Excessive deferral
- Excessive collision

The received frames are considered ”good frames”, if there are not any of the following errors:

- CRC error
- Runt frames (frames shorter than 64 bytes)
- Alignment error (in 10/100 Mbps modes only)
- Length error (non-type frames only)
- Frame size over the maximum size (for non-type frames over the maximum frame size only)
- MII_RXER input error

The maximum frame size depends on the frame type:

- The maximum size of untagged frames = 1518 bytes
- The maximum size of VLAN frames = 1522 bytes

10.5 EMAC_MTL (MAC Transaction Layer)

The MAC Transaction Layer provides FIFO memory to buffer and regulates the frames between the application system memory and the MAC. It also enables the data to be transmitted between the application clock domain and the MAC clock domains. The MTL layer has two data paths, namely the Transmit path and the Receive path. The data path for both directions is 32-bit wide and operates with a simple FIFO protocol.

10.6 PHY Interface

The DMA and the Host driver communicate through two data structures:

- Control and Status Registers (CSR)
- Descriptor lists and data buffers

For details please refer to Register Summary and Linked List Descriptors.

10.6.1 MII (Media Independent Interface)

Media Independent Interface (MII) defines the interconnection between MAC sublayers and PHYs at the data transmission rate of 10 Mbit/s and 100 Mbit/s.

10.6.1.1 Interface Signals Between MII and PHY

Interface signals between MII and PHY are shown in Figure 42.

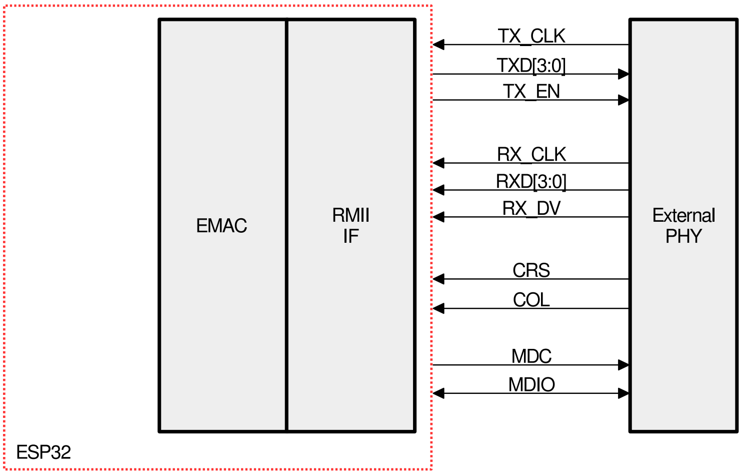


Figure 42: MII Interface

MII Interface Signal Description:

- MII_TX_CLK: TX clock signal. This signal provides the reference timing for TX data transmission. The frequencies are divided into two types: 2.5 MHz at a data transmission rate of 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_TXD[3:0]: Transmit data signal in groups of four, syn-driven by the MAC sub-layer, and valid only when the MII_TX_EN signal is valid. MII_TXD[0] is the lowest significant bit and MII_TXD[3] is the highest significant bit. When the signal MII_TX_EN is pulled low, sending data does not have any effect on the PHY.

- MII_TX_EN: Transmit data enable signal. This signal indicates that the MAC is currently sending nibbles (4 bits) for the MII. This signal must be synchronized with the first nibble of the header (MII_TX_CLK) and must be synchronized when all nibbles to be transmitted are sent to the MII.
- MII_RX_CLK: RX clock signal. This signal provides the reference timing for RX data transmission. The frequencies are divided into two types: 2.5 MHz at the data transmission rate of 10 Mbit/s, and 25 MHz at 100 Mbit/s.
- MII_RXD[3:0]: Receive data signal in groups of four, syn-driven by the PHY, and valid only when MII_RX_DV signal is valid. MII_RXD[0] is the lowest significant bit and MII_RXD[3] is the highest significant bit. When MII_RX_DV is disabled and MII_RX_ER is enabled, the specific MII_RXD[3:0] value represents specific information from the PHY.
- MII_RX_DV: Receive data valid signal. This signal indicates that the PHY is currently receiving the recovered and decoded nibble that will be transmitted to the MII. This signal must be synchronized with the first nibble of the recovered frame (MII_RX_CLK) and remain synchronized till the last nibble of the recovered frame. This signal must be disabled before the first clock cycle following the last nibble. In order to receive the frame correctly, the MII_RX_DV signal must cover the frame to be received over the time range, starting no later than when the SFD field appears.
- MII_CRS: Carrier sense signal. When the transmitting or receiving medium is in the non-idle state, the signal is enabled by the PHY. When the transmitting or receiving medium is in the idle state, the signal is disabled by the PHY. The PHY must ensure that the MII_CRS signal remains valid under conflicting conditions. This signal does not need to be synchronized with the TX and RX clocks. In full-duplex mode, this signal is insignificant.
- MII_COL: Collision detection signal. After a collision is detected on the medium, the PHY must immediately enable the collision detection signal, and the collision detection signal must remain active as long as a condition for collision exists. This signal does not need to be synchronized with the TX and RX clocks. In full-duplex mode, this signal is meaningless.
- MII_RX_ER: Receive error signal. The signal must remain for one or more cycles (MII_RX_CLK) to indicate to the MAC sublayer that an error has been detected somewhere in the frame.
- MDIO and MDC: Management Data Input/Output and Management Data Clock. The two signals constitute a serial bus defined for the Ethernet family of IEEE 802.3 standards, used to transfer control and data information to the PHY, see section Station Management Agent (SMA) Interface.

10.6.1.2 MII Clock

In MII mode, there are two directions of clock, Tx and Rx clocks in the interface between MII and the PHY. MII_TX_CLK is used to synchronize the TX data, and MII_RX_CLK is used to synchronize the RX data. The MII_RX_CLK clock is provided by the PHY. The MII_TX_CLK is provided by the chip's internal PLL or external crystal oscillator. For details regarding Figure 43, please refer to the clock-related registers in Register Summary.

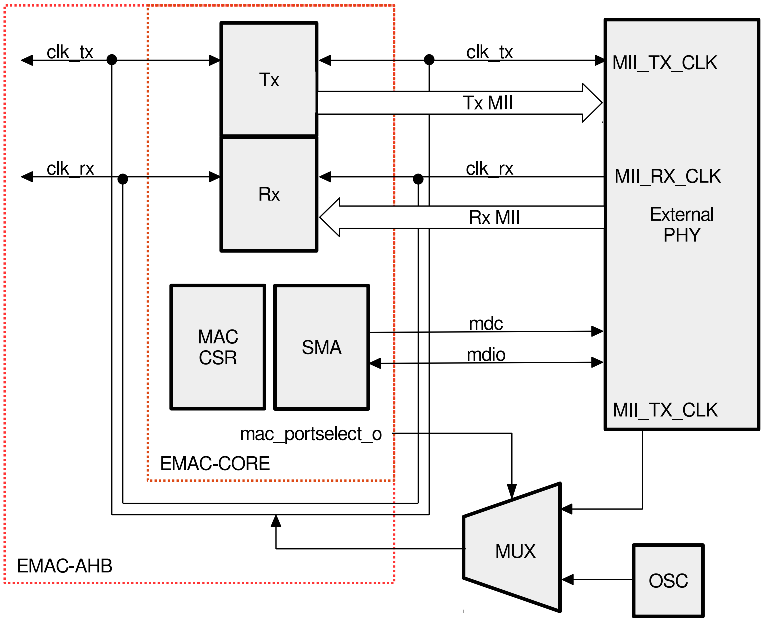


Figure 43: MII Clock

10.6.2 RMII (Reduced Media-Independent Interface)

RMII interface signals are shown in figure 44.

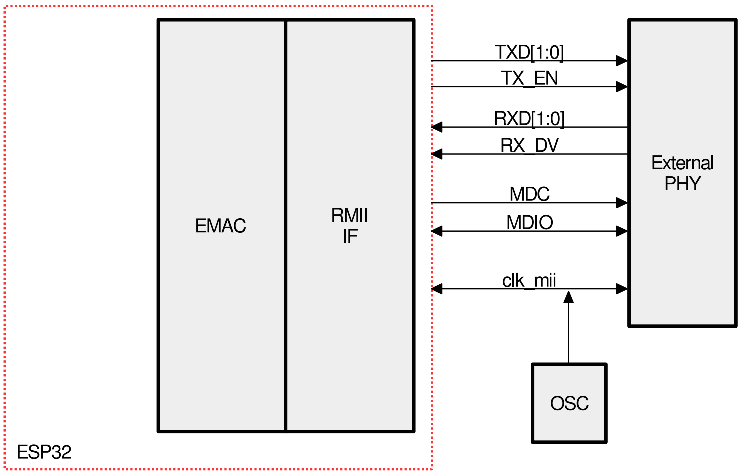


Figure 44: RMII Interface

10.6.2.1 RMII Interface Signal Description

The Reduced Media-Independent Interface (RMII) specification reduces the number of pins between the microcontroller's external peripherals and the external PHY at a data transmission rate of 10 Mbit/s or 100 Mbit/s. According to the IEEE 802.3u standard, MII includes 16 pins that contain data and control signals. The RMII specification reduces 62.5% of the pins to the number of seven.

RMII has the following features:

- Support for an operating rate of 10 Mbit/s or 100 Mbit/s
- The reference clock frequency must be 50 MHz.

- The same reference clock must be provided externally both to the MAC and the external Ethernet PHY. It provides independent 2-bit-wide Tx and Rx data paths.

10.6.2.2 RMII Clock

The configuration of the RMII clock is as figure 45 shows.

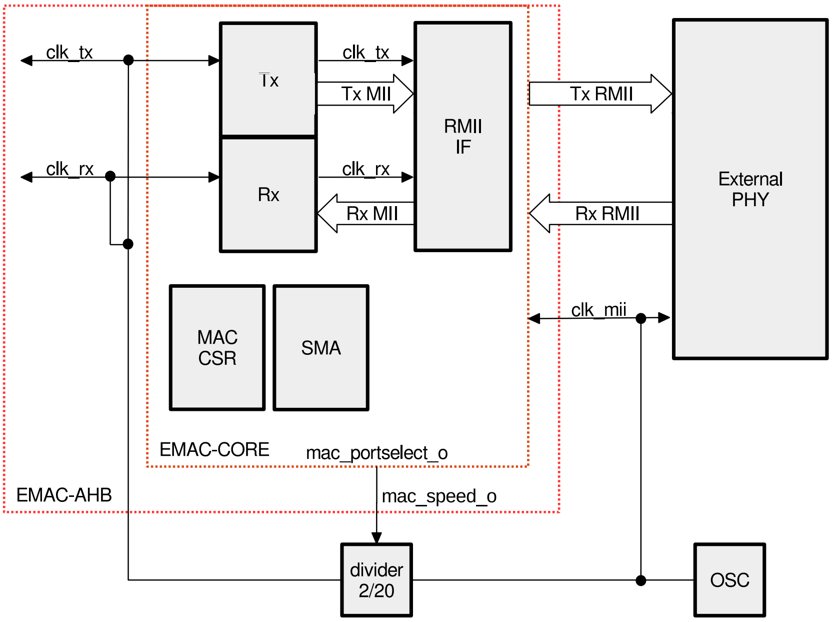


Figure 45: RMII Clock

10.6.3 Station Management Agent (SMA) Interface

As Figure 43 shows, the MAC uses MDC and MDIO signals to transfer control and data information to the PHY. The maximum clock frequency is 2.5 MHz. The clock is generated from the application clock by a clock divider. The PHY transmits register data during a write/read operation through the MDIO. This signal is driven synchronously to the MDC clock.

Please refer to Register Summary for details about the EMII Address Register and the EMII Data Register.

10.7 Ethernet DMA Features

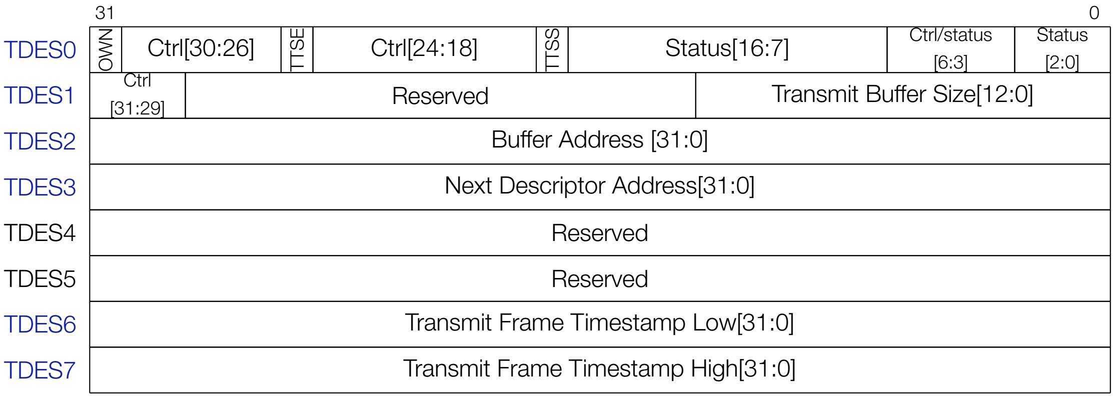
The DMA has independent Transmit and Receive engines, and a CSR (Control and Status Registers) space. The Transmit engine transfers data from the system memory to the device port (MTL), while the Receive engine transmits data from the device port to the system memory. The controller uses descriptors to efficiently move data from source to destination with minimal Host CPU intervention. The DMA is designed for packet-oriented data transmission, such as frames in Ethernet. The controller can be programmed to interrupt the Host CPU for normal situations, such as the completion of frame transmission or reception, or when errors occur.

10.8 Linked List Descriptors

This section shows the structure of the linked lists and the descriptors. Every linked list consists of eight words.

10.8.1 Transmit Descriptors

The structure of the transmitter linked lists is shown in Figure 46. Table 40 to Table 45 show the description of the linked lists.



	31							0
TDES0	OWN	Ctrl[30:26]	TTSE	Ctrl[24:18]	TTSS	Status[16:7]	Ctrl/status [6:3]	Status [2:0]
TDES1	Ctrl [31:29]	Reserved				Transmit Buffer Size[12:0]		
TDES2	Buffer Address [31:0]							
TDES3	Next Descriptor Address[31:0]							
TDES4	Reserved							
TDES5	Reserved							
TDES6	Transmit Frame Timestamp Low[31:0]							
TDES7	Transmit Frame Timestamp High[31:0]							

Figure 46: Transmit Descriptor

Table 40: Transmit Descriptor 0 (TDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit, either when it completes the frame transmission or when the buffers allocated to the descriptor are empty. The ownership bit of the First Descriptor of the frame should be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
[30]	IC: Interrupt on Completion	When set, this bit sets the Transmit Interrupt (Register 5[0]) after the present frame has been transmitted. This bit is valid only when the last segment bit (TDES0[29]) is set.
[29]	LS: Last Segment	When set, this bit indicates that the buffer contains the last segment of the frame. When this bit is set, the TBS1 or TBS2 field in TDES1 should have a non-zero value.
[28]	FS: First Segment	When set, this bit indicates that the buffer contains the first segment of a frame.
[27]	DC: Disable CRC	When this bit is set, the MAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.

Bits	Name	Description
[26]	DP: Disable Pad	When set, the MAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.
[25]	TTSE: Transmit Timestamp Enable	When set, this bit enables IEEE1588 hardware timestamping for the transmit frame referenced by the descriptor. This field is valid only when the First Segment control bit (TDES0[28]) is set.
[24]	CRCR: CRC Replacement Control	When set, the MAC replaces the last four bytes of the transmitted packet with recalculated CRC bytes. The host should ensure that the CRC bytes are present in the frame being transmitted from the Transmit Buffer. This bit is valid when the First Segment control bit (TDES0[28]) is set. In addition, CRC replacement is done only when Bit TDES0[27] is set to 1.
[23:22]	CIC: Checksum Insertion Control	These bits control the checksum calculation and insertion. The following list describes the bit encoding: • 2'b00: Checksum insertion is disabled. • 2'b01: Only IP header checksum calculation and insertion are enabled. • 2'b10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. • 2'b11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware. This field is valid when the First Segment control bit (TDES0[28]) is set.
[21]	TER: Transmit End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[20]	TCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address, rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a "don't care" value. TDES0[21] takes precedence over TDES0[20]. This bit should be set to 1.

Bits	Name	Description
[19:18]	VLIC: VLAN Insertion Control	When set, these bits request the MAC to perform VLAN tagging or untagging before transmitting the frames. If the frame is modified for VLAN tags, the MAC automatically recalculates and replaces the CRC bytes. The following list describes the values of these bits: • 2'b00: Do not add a VLAN tag. • 2'b01: Remove the VLAN tag from the frames before transmission. This option should be used only with the VLAN frames. • 2'b10: Insert a VLAN tag with the tag value programmed in VLAN Tag Inclusion or Replacement Register. • 2'b1: Replace the VLAN tag in frames with the Tag value programmed in VLAN Tag Inclusion or Replacement Register. This option should be used only with the VLAN frames.
[17]	TTSS: Transmit Timestamp Status	This field is used as a status bit to indicate that a timestamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a timestamp value captured for the transmit frame. This field is only valid when the descriptor's Last Segment control bit (TDES0[29]) is set.
[16]	IHE: IP Header Error	When set, this bit indicates that the MAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application, and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • TDES0[14]: Jabber Timeout • TDES0[13]: Frame Flush • TDES0[11]: Loss of Carrier • TDES0[10]: No Carrier • TDES0[9]: Late Collision • TDES0[8]: Excessive Collision • TDES0[2]: Excessive Deferral • TDES0[1]: Underflow Error • TDES0[16]: IP Header Error • TDES0[12]: IP Payload Error
[14]	JT: Jabber Timeout	When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit is only set when EMACCONFIG_REG's bit EMACJABBER is not set.
[13]	FF: Frame Flushed	When set, this bit indicates that the DMA or MTL flushed the frame because of a software Flush command given by the CPU.

Bits	Name	Description
[12]	IPE: IP Payload Error	When set, this bit indicates that MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application, and issues an error status in case of a mismatch.
[11]	LOC: Loss of Carrier	When set, this bit indicates that a loss of carrier occurred during frame transmission (that is, the MII_CRS signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision when the MAC operates in the half-duplex mode.
[10]	NC: No Carrier	When set, this bit indicates that the Carrier Sense signal from the PHY was not asserted during transmission.
[9]	LC: Late Collision	When set, this bit indicates that frame transmission is aborted because of a collision occurring after the collision window (64 byte-times including Preamble in MII mode, and 512 byte-times including Preamble and Carrier Extension). This bit is not valid if the Underflow Error bit is set.
[8]	EC: Excessive Collision	When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If bit EMACRETRY of EMACCONFIG_REG is set, this bit is set after the first collision, and the transmission of the frame is aborted.
[7]	VF: VLAN Frame	When set, this bit indicates that the transmitted frame is a VLAN-type frame.
[6:3]	Ctrl/status	These status bits indicate the number of collisions that occurred before the frame was transmitted. This count is not valid when the Excessive Collisions bit (TDES0[8]) is set. The core updates this status field only in the half-duplex mode.
[2]	ED: Excessive Deferral	When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (if Jumbo Frame is enabled) if bit EMACDEFERRAL of EMACCONFIG_REG is set high.
[1]	UF: Underflow Error	When set, this bit indicates that the MAC aborted the frame because the data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Bit[5] in Transmit Underflow Register (Status Register) and Bit[0] in Transmit Interrupt Register (Status Register).
[0]	DB: Deferred Bit	When set, this bit indicates that the MAC defers before transmission because of the presence of a carrier. This bit is valid only in the half-duplex mode.

Table 41: Transmit Descriptor 1 (TDES1)

Bits	Name	Description
[31:29]	SAIC: SA Insertion Control	These bits request the MAC to add or replace the Source Address field in the Ethernet frame with the value given in the MAC Address 0 register. If the Source Address field is modified in a frame, the MAC automatically recalculates and replaces the CRC bytes. The Bit[31] specifies the MAC Address Register value (1 or 0) that is used for Source Address insertion or replacement. The following list describes the values of Bits[30:29]: • 2'b00: Do not include the source address. • 2'b01: Include or insert the source address. For reliable transmission, the application must provide frames without source addresses. • 2'b10: Replace the source address. For reliable transmission, the application must provide frames with source addresses. • 2'b11: Reserved These bits are valid when the First Segment control bit (TDES0[28]) is set.
[28:16]	Reserved	Reserved
[15:13]	Reserved	Reserved
[12:0]	TBS1: Transmit Buffer 1 Size	These bits indicate the data buffer byte size in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor.

Table 42: Transmit Descriptor 2 (TDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 43: Transmit Descriptor 3 (TDES3)

Bits	Name	Description
[31:0]	Next Descriptor Address	This address contains the pointer to the physical memory where the Next Descriptor is present.

Table 44: Transmit Descriptor 6 (TDES6)

Bits	Name	Description
[31:0]	TTSL: Transmit Frame Timestamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding transmit frame. This field has the timestamp only if the Last Segment (LS) bit in the descriptor is set, and the Timestamp Status (TTSS) bit is set too.

Table 45: Transmit Descriptor 7 (TDES7)

Bits	Name	Description
[31:0]	TTSH: Transmit Frame Timestamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field has the timestamp only if the Last Segment (LS) bit in the descriptor is set, and the Timestamp Status (TTSS) bit is set too.

10.8.2 Receive Descriptors

The structure of the receiver linked lists is shown in Figure 47. Table 46 to Table 52 provide the description of the linked lists.

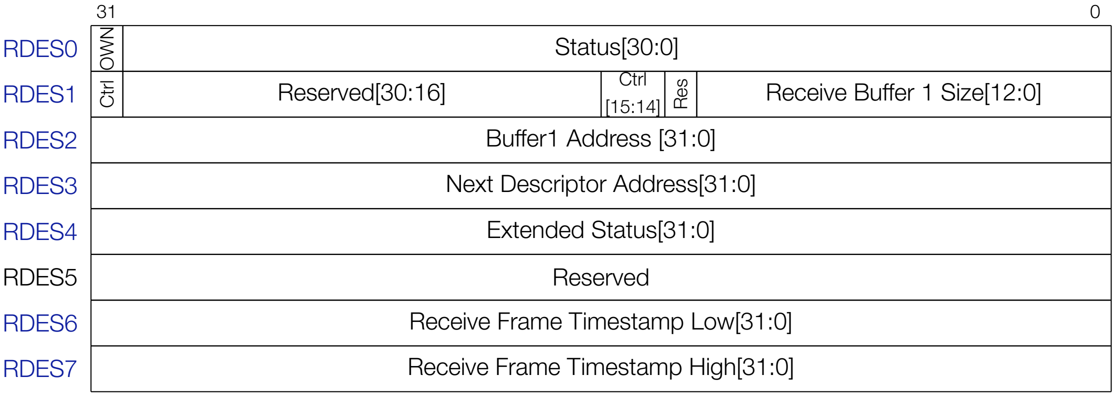


Figure 47: Receive Descriptor

Table 46: Receive Descriptor 0 (RDES0)

Bits	Name	Description
[31]	OWN: Own Bit	When set, this bit indicates that the descriptor is owned by the DMA of the DWC_gmac. When this bit is reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
[30]	AFM: Destination Address Filter Fail	When set, this bit indicates a frame that failed in the DA Filter in the MAC.
[29:16]	FL: Frame Length	These bits indicate the byte length of the received frame that was transmitted to host memory. This field is valid when Last Descriptor (RDES0[8]) is set and either the Descriptor Error (RDES0[14]) or Overflow Error bits is reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame.
[15]	ES: Error Summary	Indicates the logical OR of the following bits: • RDES0[1]: CRC Error • RDES0[3]: Receive Error • RDES0[4]: Watchdog Timeout • RDES0[6]: Late Collision • RDES0[7]: Giant Frame • RDES4[4:3]: IP Header or Payload Error • RDES0[11]: Overflow Error • RDES0[14]: Descriptor Error This field is valid only when the Last Descriptor (RDES0[8]) is set.
[14]	DE: Descriptor Error	When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when the Last Descriptor (RDES0[8]) is set.

Bits	Name	Description
[13]	SAF: Source Address Filter Fail	When set, this bit indicates that the SA field of frame failed the SA Filter in the MAC.
[12]	LE: Length Error	When set, this bit indicates that the actual length of the frame received and that the Length/Type field does not match. This bit is valid only when the Frame Type (RDES0[5]) bit is reset.
[11]	OE: Overflow Error	When set, this bit indicates that the received frame was damaged because of buffer overflow in MTL.
[10]	VLAN: VLAN Tag	When set, this bit indicates that the frame to which this descriptor is pointing is a VLAN frame tagged by the MAC. The VLAN tagging depends on checking the VLAN fields of the received frame based on the Register (VLAN Tag Register) settings.
[9]	FS: First Descriptor	When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
[8]	LS: Last Descriptor	When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame.
[7]	Timestamp Available, IP Checksum Error (Type1), or Giant Frame	When the Advanced Timestamp feature is present, and when this bit set, it indicates that a snapshot of the Timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This is valid only when the Last Descriptor bit (RDES0[8]) is set. When IP Checksum Engine (Type 1) is selected, this bit, if set, indicates one of the following: • The 16-bit IPv4 header checksum calculated by the core did not match the received checksum bytes. • The header checksum checking is bypassed for non-IPv4 frames. Otherwise, this bit, when set, indicates the Giant Frame Status. Giant frames are larger than 1,518 bytes (or 1,522 bytes for VLAN or 2,000 bytes when Bit[27] of the MAC Configuration register is set), normal frames and larger-than-9,018-byte (9,022-byte for VLAN) frames when Jumbo Frame processing is enabled.
[6]	LC: Late Collision	When set, this bit indicates that a late collision has occurred while receiving the frame in the half-duplex mode.
[5]	FT: Frame Type	When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than, or equal to, 1,536). When this bit is reset, it indicates that the received frame is an IEEE 802.3 frame. This bit is not valid for Runt frames which are less than 14 bytes.
[4]	RWT: Receive Watchdog Timeout	When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
[3]	RE: Receive Error	When set, this bit indicates that the MII_RXER signal is asserted while MII_RXDV is asserted during frame reception.

Bits	Name	Description
[2]	DE: Dribble Bit Error	When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in the MII Mode.
[1]	CE: CRC Error	When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.
[0]	Extended Status Available/ Rx MAC Address	When either Advanced Timestamp or IP Checksum Offload (Type 2) is present, this bit, when set, indicates that the extended status is available in descriptor word 4 (RDES4). This is valid only when the Last Descriptor bit (RDES0[8]) is set. This bit is invalid when Bit 30 is set. When IP Checksum Offload (Type 2) is present, this bit is set even when the IP Checksum Offload engine bypasses the processing of the received frame. The bypassing may be because of a non-IP frame or an IP frame with a non-TCP/UDP/ICMP payload. When the Advance Timestamp Feature or the IPC Full Offload is not selected, this bit indicates an Rx MAC Address status. When set, this bit indicates that the Rx MAC Address registers value (1 to 15) matched the frame's DA field. When reset, this bit indicates that the Rx MAC Address Register 0 value matched the DA field.

Table 47: Receive Descriptor 1 (RDES1)

Bits	Name	Description
[31]	Ctrl	When set, this bit prevents setting the Status Register's RI bit (CSR5[6]) for the received frame that ends in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to Host because of the RI for that frame.
[30:29]	Reserved	Reserved
[28:16]	Reserved	Reserved
[15]	RER: Receive End of Ring	When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
[14]	RCH: Second Address Chained	When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a "don't care" value. RDES1[15] takes precedence over RDES1[14].
[13]	Reserved	Reserved
[12:0]	RBS1: Receive Buffer 1 Size	Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4, even if the value of RDES2 (buffer1 address pointer) is not aligned to bus width. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor depending on the value of RCH (Bit[14]).

Table 48: Receive Descriptor 2 (RDES2)

Bits	Name	Description
[31:0]	Buffer 1 Address Pointer	These bits indicate the physical address of Buffer 1.

Table 49: Receive Descriptor 3 (RDES3)

Bits	Name	Description
[31:0]	Next Descriptor Address	This address contains the pointer to the physical memory where the Next Descriptor is present.

Table 50: Receive Descriptor 4 (RDES4)

Bits	Name	Description
[31:28]	Reserved	Reserved
[27:26]	Reserved	Reserved
[25]	Reserved	Reserved
[24]	Reserved	Reserved
[23:21]	Reserved	Reserved
[20:18]	Reserved	Reserved
[17]	Reserved	Reserved
[16]	Reserved	Reserved
[15]	Reserved	Reserved
[14]	Timestamp Dropped	When set, this bit indicates that the timestamp was captured for this frame but got dropped in the MTL Rx FIFO because of an overflow.
[13]	PTP Version	When set, this bit indicates that the received PTP message is having the IEEE 1588 version 2 format. When reset, it has the version 1 format.
[12]	PTP Frame Type	When set, this bit indicates that the PTP message is sent directly over the Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message is sent over UDP-IPv4 or UDP-IPv6. The information about IPv4 or IPv6 can be obtained from Bits 6 and 7.

Bits	Name	Description
[11:8]	Message Type	These bits are encoded to give the type of the message received. • 3'b0000: No PTP message received • 3'b0001: SYNC (all clock types) • 3'b0010: Follow_Up (all clock types) • 3'b0011: Delay_Req (all clock types) • 3'b0100: Delay_Resp (all clock types) • 3'b0101: Pdelay_Req (in peer-to-peer transparent clock) • 3'b0110: Pdelay_Resp (in peer-to-peer transparent clock) • 3'b0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) • 3'b1000: Announce • 3'b1001: Management • 3'b1010: Signaling • 3'b1011-3'b1110: Reserved • 3'b1111: PTP packet with Reserved message type
[7]	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet. This bit is updated only when Bit[10] (IPC) of Register (MAC Configuration Register) is set.
[6]	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet. This bit is updated only when Bit[10] (IPC) of Register (MAC Configuration Register) is set.
[5]	IP Checksum Bypassed	When set, this bit indicates that the checksum offload engine is bypassed.
[4]	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field. This bit is valid when either Bit 7 or Bit 6 is set.
[3]	IP Header Error	When set, this bit indicates that either the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes, or the IP datagram version is not consistent with the Ethernet Type value. This bit is valid when either Bit[7] or Bit[6] is set.
[2:0]	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the Receive Checksum Offload Engine (COE). The COE also sets these bits to 2'b00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP. • 3'b000: Unknown or did not process IP payload • 3'b001: UDP • 3'b010: TCP • 3'b011: ICMP • 3'b1xx: Reserved This bit is valid when either Bit[7] or Bit[6] is set.

Table 51: Receive Descriptor 6 (RDES6)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Timestamp Low	This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

Table 52: Receive Descriptor 7 (RDES7)

Bits	Name	Description
[31:0]	RTSH: Receive Frame Timestamp High	This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by the Last Descriptor status bit (RDES0[8]).

10.9 Register Summary

Note that specific fields or bits of a given register may have different access attributes. Below is the list of all attributes together with the abbreviations used in register descriptions.

- Read Only (RO)
- Write Only (WO)
- Read and Write (R/W)
- Read, Write, and Self Clear (R/W/SC)
- Read, Self Set, and Write Clear (R/SS/WC)
- Read, Write Set, and Self Clear (R/WS/SC)
- Read, Self Set, and Self Clear or Write Clear (R/SS/SC/WC)
- Read Only and Write Trigger (RO/WT)
- Read, Self Set, and Read Clear (R/SS/RC)
- Read, Write, and Self Update (R/W/SU)
- Latched-low (LL)
- Latched-high (LH)

Name	Description	Address	Access
DMA configuration and control registers			
DMABUSMODE_REG	Bus mode configuration	0x60029000	R/WS/SC
DMATXPOLLDEMAND_REG	Pull demand for data transmit	0x60029004	RO/WT
DMARXPOLLDEMAND_REG	Pull demand for data receive	0x60029008	RO/WT
DMARXBASEADDR_REG	Base address of the first receive descriptor	0x6002900C	R/W

Name	Description	Address	Access
DMATXBASEADDR_REG	Base address of the first transmit descriptor	0x60029010	R/W
DMASTATUS_REG	State of interrupts, errors and other events	0x60029014	R/SS/WC
DMAOPERATION_MODE_REG	Receive and Transmit operating modes and command	0x60029018	R/SS/WC
DMAIN_EN_REG	Enable / disable interrupts	0x6002901C	R/W
DMAMISSEDFR_REG	Missed Frame and Buffer Overflow Counter Register	0x60029020	R/W
DMARINTWDTIMER_REG	Watchdog timer count on receive	0x60029024	R/W
DMATXCURRDESC_REG	Pointer to current transmit descriptor	0x60029048	RO
DMARXCURRDESC_REG	Pointer to current receive descriptor	0x6002904C	RO
DMATXCURRADDR_BUF_REG	Pointer to current transmit buffer	0x60029050	RO
DMARXCURRADDR_BUF_REG	Pointer to current receive buffer	0x60029054	RO
MAC configuration and control registers			
EMACCONFIG_REG	MAC configuration	0x6002A000	R/W
EMACFF_REG	Frame filter settings	0x6002A004	R/W
EMACMIIADDR_REG	PHY configuration access	0x6002A010	R/WS/SC
EMACMIIDATA_REG	PHY data read write	0x6002A014	R/W
EMACFC_REG	frame flow control	0x6002A018	R/WS/SC(FCB) R/W(BPA)
EMACDEBUG_REG	Status debugging bits	0x6002A024	RO
PMT_RWUFFR_REG	Remote Wake-Up Frame Filter	0x6002A028	RO
PMT_CSR_REG	PMT Control and Status	0x6002A02C	RO
EMACLPI_CSR_REG	LPI Control and Status	0x6002A030	RO
EMACLPITIMERSCONTROL_REG	LPI Timers Control	0x6002A034	RO
EMACINTS_REG	Interrupt status	0x6002A038	RO
EMACINTMASK_REG	Interrupt mask	0x6002A03C	R/W
EMACADDR0HIGH_REG	Upper 16 bits of the first 6-byte MAC address	0x6002A040	R/W
EMACADDR0LOW_REG	Lower 32 bits of the first 6-byte MAC address	0x6002A044	R/W
EMACADDR1HIGH_REG	MAC address filtering and upper 16 bits of the second 6-byte MAC address	0x6002A048	R/W
EMACADDR1LOW_REG	Lower 32 bits of the second 6-byte MAC address	0x6002A04C	R/W
EMACADDR2HIGH_REG	MAC address filtering and upper 16 bits of the third 6-byte MAC address	0x6002A050	R/W
EMACADDR2LOW_REG	Lower 32 bits of the third 6-byte MAC address	0x6002A054	R/W
EMACADDR3HIGH_REG	MAC address filtering and upper 16 bits of the fourth 6-byte MAC address	0x6002A058	R/W
EMACADDR3LOW_REG	Lower 32 bits of the fourth 6-byte MAC address	0x6002A05C	R/W

Name	Description	Address	Access
EMACADDR4HIGH_REG	MAC address filtering and upper 16 bits of the fifth 6-byte MAC address	0x6002A060	R/W
EMACADDR4LOW_REG	Lower 32 bits of the fifth 6-byte MAC address	0x6002A064	R/W
EMACADDR5HIGH_REG	MAC address filtering and upper 16 bits of the sixth 6-byte MAC address	0x6002A068	R/W
EMACADDR5LOW_REG	Lower 32 bits of the sixth 6-byte MAC address	0x6002A06C	R/W
EMACADDR6HIGH_REG	MAC address filtering and upper 16 bits of the seventh 6-byte MAC address	0x6002A070	R/W
EMACADDR6LOW_REG	Lower 32 bits of the seventh 6-byte MAC address	0x6002A074	R/W
EMACADDR7HIGH_REG	MAC address filtering and upper 16 bits of the eighth 6-byte MAC address	0x6002A078	R/W
EMACADDR7LOW_REG	Lower 32 bits of the eighth 6-byte MAC address	0x6002A07C	R/W
EMACCSTATUS_REG	Link communication status	0x6002A0D8	RO
EMACWDOGTO_REG	Watchdog timeout control	0x6002A0DC	R/W
Clock configuration registers			
EMAC_EX_CLKOUT_CONF_REG	RMII clock divider setting	0x60029800	R/W
EMAC_EX_OSCCLK_CONF_REG	RMII clock half and whole divider settings	0x60029804	R/W
EMAC_EX_CLK_CTRL_REG	Clock enable and external / internal clock selection	0x60029808	R/W
PHY type and SRAM configuration registers			
EMAC_EX_PHYINF_CONF_REG	Selection of MII / RMII phy	0x6002980C	R/W
EMAC_PD_SEL_REG	Ethernet RAM power-down enable	0x60029810	R/W

10.10 Registers

Note: The value of all reset registers must be set to the reset value.

Register 10.1: DMABUSMODE_REG (0x0000)

(reserved)	DMAMIXEDBURST	DMAADDRALIBEA	PBLX8_MODE	USE_SEP_PBL	RX_DMA_PBL	FIXED_BURST	PRI_RATIO	PROG_BURST_LEN	ALT_DESC_SIZE	DESC_SKIP_LEN	DMA_ARB_SCH	SW_RST	
31 27	26	25	24	23	22 17	16	15 14	13 8	7	6 2	1	0	
0 0 0 0 0	0	0	0	0	0x01	0	0x0	0x01	0	0x00	0	1	Reset

DMAMIXEDBURST When this bit is set high and the FB(FIXES_BURST) bit is low, the AHB master interface starts all bursts of a length more than 16 with INCR (undefined burst), whereas it reverts to fixed burst transfers (INCRx and SINGLE) for burst length of 16 and less. (R/W)

DMAADDRALIBEA When this bit is set high and the FB bit is 1, the AHB interface generates all bursts aligned to the start address LS bits. If the FB bit is 0, the first burst (accessing the start address of data buffer) is not aligned, but subsequent bursts are aligned to the address. (R/W)

PBLX8_MODE When set high, this bit multiplies the programmed PBL(PROG_BURST_LEN) value (Bits[22:17] and Bits[13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, 32, 64, 128, and 256 beats depending on the PBL value. (R/W)

USE_SEP_PBL When set high, this bit configures the Rx DMA to use the value configured in Bits[22:17] as PBL. The PBL value in Bits[13:8] is applicable only to the Tx DMA operations. When reset to low, the PBL value in Bits[13:8] is applicable for both DMA engines. (R/W)

RX_DMA_PBL This field indicates the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read or Write.The Rx DMA always attempts to burst as specified in the RPBL(RX_DMA_PBL) bit each time it starts a burst transfer on the host bus. You can program RPBL with values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. This field is valid and applicable only when USP(USE_SEP_PBL) is set high. (R/W)

FIXED_BURST This bit controls whether the AHB master interface performs fixed burst transfers or not. When set, the AHB interface uses only SINGLE, INCR4, INCR8, or INCR16 during start of the normal burst transfers. When reset, the AHB interface uses SINGLE and INCR burst transfer operations. (R/W)

PRI_RATIO These bits control the priority ratio in the weighted round-robin arbitration between the Rx DMA and Tx DMA. These bits are valid only when Bit 1 (DA) is reset. The priority ratio Rx:Tx represented by each bit: (R/W)

- 2'b00 — 1: 1
- 2'b01 — 2: 0
- 2'b10 — 3: 1
- 2'b11 — 4: 1

Continued on the next page...

Register 10.1: DMABUSMODE_REG (0x0000)

Continued from the previous page ...

PROG_BURST_LEN These bits indicate the maximum number of beats to be transferred in one DMA transaction. If the number of beats to be transferred is more than 32, then perform the following steps: 1. Set the PBLx8 mode; 2. Set the PBL. (R/W)

ALT_DESC_SIZE When set, the size of the alternate descriptor increases to 32 bytes. (R/W)

DESC_SKIP_LEN This bit specifies the number of Word to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When the DSL(DESC_SKIP_LEN) value is equal to zero, the descriptor table is taken as contiguous by the DMA in Ring mode. (R/W)

DMA_ARB_SCH This bit specifies the arbitration scheme between the transmit and receive paths. 1'b0: weighted round-robin with RX: TX or TX: RX, priority specified in PR (bit[15:14]); 1'b1 Fixed priority (Rx priority to Tx). (R/W)

SW_RST When this bit is set, the MAC DMA Controller resets the logic and all internal registers of the MAC. It is cleared automatically after the reset operation is complete in all of the ETH_MAC clock domains. Before reprogramming any register of the ETH_MAC, you should read a zero (0) value in this bit. (R/WS/SC)

Register 10.2: DMATXPOLLDEMAND_REG (0x0004)

31	0	
0x000000000		Reset

TRANS_POLL_DEMAND When these bits are written with any value, the DMA reads the current descriptor to which the Register (Current Host Transmit Descriptor Register) is pointing. If that descriptor is not available (owned by the Host), the transmission returns to the suspend state and Bit[2] (TU) of Status Register is asserted. If the descriptor is available, the transmission resumes. (RO/WT)

Register 10.3: DMARXPOLLDEMAND_REG (0x0008)

31	0	
0x000000000		Reset

RECV_POLL_DEMAND When these bits are written with any value, the DMA reads the current descriptor to which the Current Host Receive Descriptor Register is pointing. If that descriptor is not available (owned by the Host), the reception returns to the Suspended state and Bit[7] (RU) of Status Register is asserted. If the descriptor is available, the Rx DMA returns to the active state. (RO/WT)

Register 10.4: DMARXBASEADDR_REG (0x000C)

31 0	
0x000000000	Reset

START_RECV_LIST This field contains the base address of the first descriptor in the Receive Descriptor list. The LSB Bits[1:0] are ignored and internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only. (R/W)

Register 10.5: DMATXBASEADDR_REG (0x0010)

31 0	
0x000000000	Reset

START_TRANS_LIST This field contains the base address of the first descriptor in the Transmit Descriptor list. The LSB Bits[1:0] are ignored and are internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only. (R/W)

Register 10.6: DMASTATUS_REG (0x0014)

(reserved)	TS_TRI_INT	EMAC_PMT_INT	(reserved)	ERROR_BITS	TRANS_PROC_STATE	RECV_PROC_STATE	NORM_INT_SUMM	ABN_INT_SUMM	EARLY_RECV_INT	FATAL_BUS_ERR_INT	(reserved)	EARLY_TRANS_INT	RECV_WDT_TO	RECV_PROC_STOP	RECV_BUF_UNAVAIL	RECV_INT	TRANS_UNDFLOW	RECV_OVFLOW	TRANS_JABBER_TO	TRANS_BUF_UNAVAIL	TRANS_PROC_STOP	TRANS_INT	
31 30	29	28	27 26	25 23	22 20	19 17	16	15	14	13	12 11	10	9	8	7	6	5	4	3	2	1	0	
0 0	0	0	0 0	0x0	0x0	0x0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	Reset

TS_TRI_INT This bit indicates an interrupt event in the Timestamp Generator block of the ETH_MAC. The software must read the corresponding registers in the ETH_MAC to get the exact cause of the interrupt and clear its source to reset this bit to 1'b0. (RO)

EMAC_PMT_INT This bit indicates an interrupt event in the PMT module of the ETH_MAC. The software must read the PMT Control and Status Register in the MAC to get the exact cause of interrupt and clear its source to reset this bit to 1'b0. (RO)

Continued on the next page...

Register 10.6: DMASTATUS_REG (0x0014)

Continued from the previous page ...

ERROR_BITS This field indicates the type of error that caused a Bus Error, for example, error response on the AHB interface. This field is valid only when Bit[13] (FBI) is set. This field does not generate an interrupt. (RO)

- 3'b000: Error during Rx DMA Write Data Transfer.
- 3'b011: Error during Tx DMA Read Data Transfer.
- 3'b100: Error during Rx DMA Descriptor Write Access.
- 3'b101: Error during Tx DMA Descriptor Write Access.
- 3'b110: Error during Rx DMA Descriptor Read Access.
- 3'b111: Error during Tx DMA Descriptor Read Access.

TRANS_PROC_STATE This field indicates the Transmit DMA FSM state. This field does not generate an interrupt. (RO)

- 3'b000: Stopped. Reset or Stop Transmit Command issued.
- 3'b001: Running. Fetching Transmit Transfer Descriptor.
- 3'b010: Reserved for future use.
- 3'b011: Running. Waiting for TX packets.
- 3'b100: Suspended. Receive Descriptor Unavailable.
- 3'b101: Running. Closing Transmit Descriptor.
- 3'b110: TIME_STAMP write state.
- 3'b111: Running. Transferring the TX packets data from transmit buffer to host memory.

RECV_PROC_STATE This field indicates the Receive DMA FSM state. This field does not generate an interrupt. (RO)

- 3'b000: Stopped. Reset or Stop Receive Command issued.
- 3'b001: Running. Fetching Receive Transfer Descriptor.
- 3'b010: Reserved for future use.
- 3'b011: Running. Waiting for RX packets.
- 3'b100: Suspended. Receive Descriptor Unavailable.
- 3'b101: Running. Closing Receive Descriptor.
- 3'b110: TIME_STAMP write state.
- 3'b111: Running. Transferring the TX packets data from receive buffer to host memory.

Continued on the next page...

Register 10.6: DMASTATUS_REG (0x0014)

Continued from the previous page ...

NORM_INT_SUMM Normal Interrupt Summary bit value is the logical OR of the following bits when the corresponding interrupt bits are enabled in Interrupt Enable Register:(R/SS/WC)

- Bit[0]: Transmit Interrupt.
- Bit[2]: Transmit Buffer Unavailable.
- Bit[6]: Receive Interrupt.
- Bit[14]: Early Receive Interrupt. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes NIS to be set, is cleared.

ABN_INT_SUMM Abnormal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register: (R/SS/WC)

- Bit[1]: Transmit Process Stopped.
- Bit[3]: Transmit Jabber Timeout.
- Bit[4]: Receive FIFO Overflow.
- Bit[5]: Transmit Underflow.
- Bit[7]: Receive Buffer Unavailable. Bit[8]: Receive Process Stopped.
- Bit[9]: Receive Watchdog Timeout.
- Bit[10]: Early Transmit Interrupt.
- Bit[13]: Fatal Bus Error. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes AIS to be set, is cleared.

EARLY_RECV_INT This bit indicates that the DMA filled the first data buffer of the packet. This bit is cleared when the software writes 1 to this bit or when Bit[6] (RI) of this register is set (whichever occurs earlier). (R/SS/WC)

FATAL_BUS_ERR_INT This bit indicates that a bus error occurred, as described in Bits [25:23]. When this bit is set, the corresponding DMA engine disables all of its bus accesses. (R/SS/WC)

EARLY_TRANS_INT This bit indicates that the frame to be transmitted is fully transferred to the MTL Transmit FIFO. (R/SS/WC)

RECV_WDT_TO When set, this bit indicates that the Receive Watchdog Timer expired while receiving the current frame and the current frame is truncated after the watchdog timeout. (R/SS/WC)

Continued on the next page...

Register 10.6: DMASTATUS_REG (0x0014)

Continued from the previous page ...

RECV_PROC_STOP This bit is asserted when the Receive Process enters the Stopped state. (R/SS/WC)

RECV_BUF_UNAVAIL This bit indicates that the host owns the Next Descriptor in the Receive List and the DMA cannot acquire it. The Receive Process is suspended. To resume processing Receive descriptors, the host should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, the Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor is owned by the DMA. (R/SS/WC)

RECV_INT This bit indicates that the frame reception is complete. When reception is complete, the Bit[31] of RDES1 (Disable Interrupt on Completion) is reset in the last Descriptor, and the specific frame status information is updated in the descriptor. The reception remains in the Running state. (R/SS/WC)

TRANS_UNDFLOW This bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set. (R/SS/WC)

RECV_OVFLOW This bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to the application, the overflow status is set in RDES0[11]. (R/SS/WC)

TRANS_JABBER_TO This bit indicates that the Transmit Jabber Timer expired, which happens when the frame size exceeds 2,048 (10,240 bytes when the Jumbo frame is enabled). When the Jabber Timeout occurs, the transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert. (R/SS/WC)

TRANS_BUF_UNAVAIL This bit indicates that the host owns the Next Descriptor in the Transmit List and the DMA cannot acquire it. Transmission is suspended. Bits[22:20] explain the Transmit Process state transitions. To resume processing Transmit descriptors, the host should change the ownership of the descriptor by setting TDES0[31] and then issue a Transmit Poll Demand command. (R/SS/WC)

TRANS_PROC_STOP This bit is set when the transmission is stopped. (R/SS/WC)

TRANS_INT This bit indicates that the frame transmission is complete. When transmission is complete, Bit[31] (OWN) of TDES0 is reset, and the specific frame status information is updated in the descriptor. (R/SS/WC)

Register 10.7: DMAOPERATION_MODE_REG (0x0018)

(reserved)	DIS_DROP_TCPIP_ERR_FRAM	RX_STORE_FORWARD	DIS_FLUSH_RECV_FRAMES	(reserved)	(reserved)	TX_STR_FWD	FLUSH_TX_FIFO	(reserved)	TX_THRESH_CTRL	START_STOP_TRANSMISSION_COMMAND	(reserved)	(reserved)	(reserved)	FWD_ERR_FRAME	FWD_UNDER_GF	DROP_GFRM	RX_THRESH_CTRL	OPT_SECOND_FRAME	START_STOP_RX	(reserved)	
31 27	26	25	24	23	22	21	20	19 17	16 14	13	12 11	10 9	8	7	6	5	4 3	2	1	0	
0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

DIS_DROP_TCPIP_ERR_FRAM When this bit is set, the MAC does not drop the frames which only have errors detected by the Receive Checksum engine.When this bit is reset, all error frames are dropped if the Fwd_Err_Frame bit is reset. (R/W)

RX_STORE_FORWARD When this bit is set, the MTL reads a frame from the Rx FIFO only after the complete frame has been written to it. (R/W)

DIS_FLUSH_RECV_FRAMES When this bit is set, the Rx DMA does not flush any frames because of the unavailability of receive descriptors or buffers. (R/W)

TX_STR_FWD When this bit is set, transmission starts when a full frame resides in the MTL Transmit FIFO. When this bit is set, the TX_THRESH_CTRL values specified in TX_THRESH_CTRL are ignored. (R/W)

FLUSH_TX_FIFO When this bit is set, the transmit FIFO controller logic is reset to its default values and thus all data in the Tx FIFO is lost or flushed. This bit is cleared internally when the flushing operation is complete. (R/WS/SC)

TX_THRESH_CTRL These bits control the threshold level of the MTL Transmit FIFO. Transmission starts when the frame size within the MTL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when TX_STR_FWD is reset. 3'b000: 64, 3'b001: 128, 3'b010: 192, 3'b011: 256, 3'b100: 40, 3'b101: 32, 3'b110: 24, 3'b111: 16. (R/W)

START_STOP_TRANSMISSION_COMMAND When this bit is set, transmission is placed in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. (R/W)

FWD_ERR_FRAME When this bit is reset, the Rx FIFO drops frames with error status (CRC error, collision error, giant frame, watchdog timeout, or overflow). (R/W)

FWD_UNDER_GF When set, the Rx FIFO forwards Undersized frames (that is, frames with no Error and length less than 64 bytes) including pad-bytes and CRC.

DROP_GFRM When set, the MAC drops the received giant frames in the Rx FIFO, that is, frames that are larger than the computed giant frame limit. (R/W)

Continued on the next page...

Register 10.7: DMAOPERATION_MODE_REG (0x0018)

Continued from the previous page ...

RX_THRESH_CTRL These two bits control the threshold level of the MTL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MTL Receive FIFO is larger than the threshold. 2'b00: 64; 2'b01: 32; 2'b10: 96; 2'b11: 128. (R/W)

OPT_SECOND_FRAME When this bit is set, it instructs the DMA to process the second frame of the Transmit data even before the status for the first frame is obtained. (R/W)

START_STOP_RX When this bit is set, the Receive process is placed in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes the incoming frames.When this bit is cleared, the Rx DMA operation is stopped after the transfer of the current frame. (R/W)

Register 10.8: DMAIN_EN_REG (0x001C)

(reserved)	DMAIN_NISE	DMAIN_AISE	DMAIN_ERIE	DMAIN_FBEE	(reserved)	DMAIN_ETIE	DMAIN_RWTE	DMAIN_RSE	DMAIN_RBUE	DMAIN_RIE	DMAIN_UIE	DMAIN_OIE	DMAIN_TJTE	DMAIN_TBUE	DMAIN_TSE	DMAIN_TIE	
31 17	16	15	14	13	12 11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	Reset

DMAIN_NISE When this bit is set, normal interrupt summary is enabled. When this bit is reset, normal interrupt summary is disabled. This bit enables the following interrupts in Status Register: (R/W)

- Bit[0]: Transmit Interrupt.
- Bit[2]: Transmit Buffer Unavailable.
- Bit[6]: Receive Interrupt.
- Bit[14]: Early Receive Interrupt.

DMAIN_AISE When this bit is set, abnormal interrupt summary is enabled. When this bit is reset, the abnormal interrupt summary is disabled. This bit enables the following interrupts in Status Register:(R/W)

- Bit[1]: Transmit Process Stopped.
- Bit[3]: Transmit Jabber Timeout.
- Bit[4]: Receive Overflow.
- Bit[5]: Transmit Underflow.
- Bit[7]: Receive Buffer Unavailable.
- Bit[8]: Receive Process Stopped.
- Bit[9]: Receive Watchdog Timeout.
- Bit[10]: Early Transmit Interrupt.
- Bit[13]: Fatal Bus Error.

DMAIN_ERIE When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Early Receive Interrupt is enabled. When this bit is reset, the Early Receive Interrupt is disabled. (R/W)

Continued on the next page...

Register 10.8: DMAIN_EN_REG (0x001C)

Continued from the previous page ...

DMAIN_FBEE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Fatal Bus Error Interrupt is enabled. When this bit is reset, the Fatal Bus Error Enable Interrupt is disabled. (R/W)

DMAIN_ETIE When this bit is set with an Abnormal Interrupt Summary Enable (Bit[15]), the Early Transmit Interrupt is enabled. When this bit is reset, the Early Transmit Interrupt is disabled. (R/W)

DMAIN_RWTE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Watchdog Timeout Interrupt is enabled. When this bit is reset, the Receive Watchdog Timeout Interrupt is disabled. (R/W)

DMAIN_RSE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Stopped Interrupt is enabled. When this bit is reset, the Receive Stopped Interrupt is disabled. (R/W)

DMAIN_RBUE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Buffer Unavailable Interrupt is enabled. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled. (R/W)

DMAIN_RIE When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Receive Interrupt is enabled. When this bit is reset, the Receive Interrupt is disabled. (R/W)

DMAIN_UIE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmit Underflow Interrupt is enabled. When this bit is reset, the Underflow Interrupt is disabled. (R/W)

DMAIN_OIE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Receive Overflow Interrupt is enabled. When this bit is reset, the Overflow Interrupt is disabled. (R/W)

DMAIN_TJTE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmit Jabber Timeout Interrupt is enabled. When this bit is reset, the Transmit Jabber Timeout Interrupt is disabled. (R/W)

DMAIN_TBUE When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Transmit Buffer Unavailable Interrupt is enabled. When this bit is reset, the Transmit Buffer Unavailable Interrupt is disabled. (R/W)

DMAIN_TSE When this bit is set with Abnormal Interrupt Summary Enable (Bit[15]), the Transmission Stopped Interrupt is enabled. When this bit is reset, the Transmission Stopped Interrupt is disabled. (R/W)

DMAIN_TIE When this bit is set with Normal Interrupt Summary Enable (Bit[16]), the Transmit Interrupt is enabled. When this bit is reset, the Transmit Interrupt is disabled. (R/W)

Register 10.9: DMAMISSEDFR_REG (0x0020)

(reserved)		Overflow_BFOC	Overflow_FC		Overflow_BMFC	Missed_FC		
30	29	28	27	17	16	10	0	
0	0	0x0	0x0		0x0	0x0		Reset

Overflow_BFOC This bit is set every time the Overflow Frame Counter (Bits[27:17]) overflows, that is, the Rx FIFO overflows with the overflow frame counter at maximum value. In such a scenario, the overflow frame counter is reset to all-zeros and this bit indicates that the rollover happened. (R/SS/RC)

Overflow_FC This field indicates the number of frames missed by the application. This counter is incremented each time the MTL FIFO overflows. The counter is cleared when this register is read. (R/SS/RC)

Overflow_BMFC This bit is set every time Missed Frame Counter (Bits[15:0]) overflows, that is, the DMA discards an incoming frame because of the Host Receive Buffer being unavailable with the missed frame counter at maximum value. In such a scenario, the Missed frame counter is reset to all-zeros and this bit indicates that the rollover happened. (R/SS/RC)

Missed_FC This field indicates the number of frames missed by the controller because of the Host Receive Buffer being unavailable. This counter is incremented each time the DMA discards an incoming frame. The counter is cleared when this register is read. (R/SS/RC)

Register 10.10: DMARINTWDTIMER_REG (0x0024)

(reserved)		RIWTC		
31	8	7	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0x000		Reset

RIWTC This bit indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI (RECV_INT) status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watchdog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when the RI bit is set high because of automatic setting of RI as per RDES1[31] of any received frame. (R/W)

Register 10.11: DMATXCURRDESC_REG (0x0048)

31	0	
0x000000000		Reset

TRANS_DSCR_ADDR_PTR The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

Register 10.12: DMARXCURRDESC_REG (0x004C)

31	0	
0x000000000		Reset

RECV_DSCR_ADDR_PTR The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

Register 10.13: DMATXCURRADDR_BUF_REG (0x0050)

31	0	
0x000000000		Reset

TRANS_BUFF_ADDR_PTR The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

Register 10.14: DMARXCURRADDR_BUF_REG (0x0054)

31	0	
0x000000000		Reset

RECV_BUFF_ADDR_PTR The address of the current receive descriptor list. Cleared on Reset. Pointer updated by the DMA during operation. (RO)

Register 10.15: EMACCONFIG_REG (0x1000)

(reserved)	SAIRC	ASS2KP	(reserved)	EMACWATCHDOG	EMACJABBER	(reserved)	EMACJUMBOFRAME	EMACINTERFRAMEGAP	EMACDISABLECRS	EMACMII	EMACFESPEED	EMACRXOWN	EMACLOOPBACK	EMACDUPLEX	EMACRXIPCOFFLOAD	EMACRETRY	(reserved)	EMACPADCRCSTRIP	EMACBACKOFFLIMIT	EMACDEFERRALCHECK	EMACTX	EMACRX	PLTF	
31	30 28	27	26 24	23	22	21	20	19 17	16	15	14	13	12	11	10	9	8	7	6 5	4	3	2	1 0	
0	0x0	0	0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	0	0	0	0x0	Reset

SAIRC This field controls the source address insertion or replacement for all transmitted frames. Bit[30] specifies which MAC Address register (0 or 1) is used for source address insertion or replacement based on the values of Bits [29:28]: (R/W)

- 2'b0x: The input signals mti_sa_ctrl_i and ati_sa_ctrl_i control the SA field generation.
- 2'b10: If Bit[30] is set to 0, the MAC inserts the content of the MAC Address 0 registers in the SA field of all transmitted frames. If Bit[30] is set to 1 the MAC inserts the content of the MAC Address 1 registers in the SA field of all transmitted frames.
- 2'b11: If Bit[30] is set to 0, the MAC replaces the content of the MAC Address 0 registers in the SA field of all transmitted frames. If Bit[30] is set to 1, the MAC replaces the content of the MAC Address 1 registers in the SA field of all transmitted frames.

ASS2KP When set, the MAC considers all frames, with up to 2,000 bytes length, as normal packets. When Bit[20] (JE) is not set, the MAC considers all received frames of size more than 2K bytes as Giant frames. When this bit is reset and Bit[20] (JE) is not set, the MAC considers all received frames of size more than 1,518 bytes (1,522 bytes for tagged) as Giant frames. When Bit[20] is set, setting this bit has no effect on Giant Frame status. (R/W)

EMACWATCHDOG When this bit is set, the MAC disables the watchdog timer on the receiver. The MAC can receive frames of up to 16,383 bytes. When this bit is reset, the MAC does not allow a receive frame which more than 2,048 bytes (10,240 if JE is set high) or the value programmed in Register (Watchdog Timeout Register). The MAC cuts off any bytes received after the watchdog limit number of bytes. (R/W)

EMACJABBER When this bit is set, the MAC disables the jabber timer on the transmitter. The MAC can transfer frames of up to 16,383 bytes. When this bit is reset, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if JE is set high) during transmission. (R/W)

EMACJUMBOFRAME When this bit is set, the MAC allows Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames) without reporting a giant frame error in the receive frame status. (R/W)

Continued on the next page...

Register 10.15: EMACCONFIG_REG (0x1000)

Continued from the previous page ...

EMACINTERFRAMEGAP These bits control the minimum IFG between frames during transmission. (R/W)

- 3'b000: 96 bit times.
- 3'b001: 88 bit times.
- 3'b010: 80 bit times.
- 3'b111: 40 bit times. In the half-duplex mode, the minimum IFG can be configured only for 64 bit times (IFG = 100). Lower values are not considered.

EMACDISABLECRS When set high, this bit makes the MAC transmitter ignore the MII CRS signal during frame transmission in the half-duplex mode. This request results in no errors generated because of Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors because of Carrier Sense and can even abort the transmissions. (R/W)

EMACMII This bit selects the Ethernet line speed. It should be set to 1 for 10 or 100 Mbps operations. In 10 or 100 Mbps operations, this bit, along with FES(EMACFESPEED) bit, it selects the exact linespeed. In the 10/100 Mbps-only operations, the bit is always 1. (R/W)

EMACFESPEED This bit selects the speed in the MII, RMII interface. 0: 10 Mbps; 1: 100 Mbps. (R/W)

EMACRXOWN When this bit is set, the MAC disables the reception of frames when the TX_EN is asserted in the half-duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in the full-duplex mode. (R/W)

EMACLOOPBACK When this bit is set, the MAC operates in the loopback mode MII. The MII Receive clock input (CLK_RX) is required for the loopback to work properly, because the transmit clock is not looped-back internally. (R/W)

EMACDUPLEX When this bit is set, the MAC operates in the full-duplex mode where it can transmit and receive simultaneously. This bit is read only with default value of 1'b1 in the full-duplex-mode. (R/W)

EMACRXIPCOFFLOAD When this bit is set, the MAC calculates the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25/26 or 29/30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The MAC also appends the 16-bit checksum calculated for the IP header datagram payload (bytes after the IPv4 header) and appends it to the Ethernet frame transferred to the application (when Type 2 COE is deselected). When this bit is reset, this function is disabled. (R/W)

Continued on the next page...

Register 10.15: EMACCONFIG_REG (0x1000)

Continued from the previous page ...

EMACRETRY When this bit is set, the MAC attempts only one transmission. When a collision occurs on the MII interface, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status. When this bit is reset, the MAC attempts retries based on the settings of the BL field (Bits [6:5]). This bit is applicable only in the half-duplex mode. (R/W)

EMACPADCRCSTRIP When this bit is set, the MAC strips the Pad or FCS field on the incoming frames only if the value of the length field is less than 1,536 bytes. All received frames with length field greater than or equal to 1,536 bytes are passed to the application without stripping the Pad or FCS field. When this bit is reset, the MAC passes all incoming frames, without modifying them, to the Host. (R/W)

EMACBACKOFFLIMIT The Back-Off limit determines the random integer number (r) of slot time delays (512 bit times for 10/100 Mbps) for which the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only in the half-duplex mode.

- 00: k= min (n, 10).
- 01: k = min (n, 8).
- 10: k = min (n, 4).
- 11: k = min (n, 1), n = retransmission attempt. The random integer r takes the value in the range 0 ~ 2000.

EMACDEFERRALCHECK Deferral Check. (R/W)

EMACTX When this bit is set, the transmit state machine of the MAC is enabled for transmission on the MII. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and does not transmit any further frames. (R/W)

EMACRX When this bit is set, the receiver state machine of the MAC is enabled for receiving frames from the MII. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames from the MII. (R/W)

PLTF These bits control the number of preamble bytes that are added to the beginning of every Transmit frame. The preamble reduction occurs only when the MAC is operating in the full-duplex mode. 2’b00: 7 bytes of preamble. 2’b01: 5 bytes of preamble. 2’b10: 3 bytes of preamble. (R/W)

Register 10.16: EMACFF_REG (0x1004)

RECEIVE_ALL	(reserved)	SAFE	SAIF	PCF	DBF	PAM	DAIF	(reserved)	PMODE	
31	30 10	9	8	7 6	5	4	3	2 1	0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0x0	0	0	0	0 0	0	Reset

RECEIVE_ALL When this bit is set, the MAC Receiver module passes all received frames, irrespective of whether they pass the address filter or not, to the Application. The result of the SA or DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes only those frames to the Application that pass the SA or DA address filter. (R/W)

SAFE When this bit is set, the MAC compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison fails, the MAC drops the frame. When this bit is reset, the MAC forwards the received frame to the application with updated SAF bit of the Rx Status depending on the SA address comparison. (R/W)

SAIF When this bit is set, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA Address filter. When this bit is reset, frames whose SA does not match the SA registers are marked as failing the SA Address filter. (R/W)

PCF These bits control the forwarding of all control frames (including unicast and multicast Pause frames). (R/W)

- 2'b00: MAC filters all control frames from reaching the application.
- 2'b01: MAC forwards all control frames except Pause frames to application even if they fail the Address filter.
- 2'b10: MAC forwards all control frames to application even if they fail the Address Filter.
- 2'b11: MAC forwards control frames that pass the Address Filter.

The following conditions should be true for the Pause frames processing:

- Condition 1: The MAC is in the full-duplex mode and flow control is enabled by setting Bit 2 (RFE) of Register (Flow Control Register) to 1.
- Condition 2: The destination address (DA) of the received frame matches the special multicast address or the MAC Address 0 when Bit 3 (UP) of the Register(Flow Control Register) is set.
- Condition 3: The Type field of the received frame is 0x8808 and the OPCODE field is 0x0001.

DBF When this bit is set, the AFM(Address Filtering Module) module blocks all incoming broadcast frames. In addition, it overrides all other filter settings. When this bit is reset, the AFM module passes all received broadcast frames. (R/W)

PAM When set, this bit indicates that all received frames with a multicast destination address (first bit in the destination address field is '1') are passed. (R/W)

Continued on the next page...

Register 10.16: EMACFF_REG (0x1004)

Continued from the previous page ...

DAIF When this bit is set, the Address Check block operates in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When reset, normal filtering of frames is performed. (R/W)

PMODE When this bit is set, the Address Filter module passes all incoming frames irrespective of the destination or source address. The SA or DA Filter Fails status bits of the Receive Status Word are always cleared when PR(PRT$_RATIO)isset.(R/W)$

Register 10.17: EMACMIIADDR_REG (0x1010)

(reserved)	MIIDEV	MIIREG	MIICSRCLK	MIIWRITE	MIIBUSY	
31 16	15 11	10 6	5 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	0x00	0x00	0	0	Reset

MIIDEV This field indicates which of the 32 possible PHY devices are being accessed. (R/W)

MIIREG These bits select the desired MII register in the selected PHY device. (R/W)

MIICSRCLK CSR clock range: 1.0 MHz ~ 2.5 MHz. (R/W)

- 4'b0000: When the APB clock frequency is 80 MHz, the MDC clock frequency is APB CLK/42;
- 4'b0000: When the APB clock frequency is 40 MHz, the MDC clock frequency is APB CLK/26.

MIIWRITE When set, this bit indicates to the PHY that this is a Write operation using the MII Data register. If this bit is not set, it indicates that this is a Read operation, that is, placing the data in the MII Data register. (R/W)

MIIBUSY This bit should read logic 0 before writing to PHY Addr Register and PHY data Register. During a PHY register access, the software sets this bit to 1'b1 to indicate that a Read or Write access is in progress. PHY data Register is invalid until this bit is cleared by the MAC. Therefore, PHY data Register (MII Data) should be kept valid until the MAC clears this bit during a PHY Write operation. Similarly for a read operation, the contents of Register 5 are not valid until this bit is cleared. The subsequent read or write operation should happen only after the previous operation is complete. Because there is no acknowledgment from the PHY to MAC after a read or write operation is completed, there is no change in the functionality of this bit even when the PHY is not present. (R/WS/SC)

Register 10.18: EMACMIIDATA_REG (0x1014)

(reserved)	MII_DATA	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00000	Reset

MII_DATA This field contains the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation. (R/W)

Register 10.19: EMACFC_REG (0x1018)

PAUSE_TIME	(reserved)	PLT	UPFD	RFCE	TFCE	FCBBA	
31 16	15 6	5 4	3	2	1	0	
0x00000	0 0 0 0 0 0 0 0 0 0	0x0	0	0	0	0	Reset

PAUSE_TIME This field holds the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the MII clock domain, then consecutive writes to this register should be performed only after at least four clock cycles in the destination clock domain. (R/W)

PLT This field configures the threshold of the Pause timer automatic retransmission of the Pause frame. The threshold values should be always less than the Pause Time configured in Bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second Pause frame is automatically transmitted at 228 (256-28) slot times after the first Pause frame is transmitted. The following list provides the threshold values for different values: (R/W)

- 2'b00: The threshold is Pause time minus 4 slot times (PT-4 slot times).
- 2'b01: The threshold is Pause time minus 28 slot times (PT-28 slot times).
- 2'b10: The threshold is Pause time minus 144 slot times (PT-144 slot times).
- 2'b11: The threshold is Pause time minus 256 slot times (PT-256 slot times). The slot time is defined as the time taken to transmit 512 bits (64 bytes) on the MII interface.

UPFD A pause frame is processed when it has the unique multicast address specified in the IEEE Std 802.3. When this bit is set, the MAC can also detect Pause frames with unicast address of the station. This unicast address should be as specified in the EMACADDR0 High Register and EMACADDR0 Low Register. When this bit is reset, the MAC only detects Pause frames with unique multicast address. (R/W)

RFCE When this bit is set, the MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time. When this bit is reset, the decode function of the Pause frame is disabled. (R/W)

TFCE In the full-duplex mode, when this bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In the half-duplex mode, when this bit is set, the MAC enables the backpressure operation. When this bit is reset, the backpressure feature is disabled. (R/W)

Continued on the next page...

Register 10.19: EMACFC_REG (0x1018)

Continued from the previous page ...

FCBBA This bit initiates a Pause frame in the full-duplex mode and activates the backpressure function in the half-duplex mode if the TFCE bit is set. In the full-duplex mode, this bit should be read as 1'b0 before writing to the Flow Control register. To initiate a Pause frame, the Application must set this bit to 1'b1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause frame transmission, the MAC resets this bit to 1'b0. The Flow Control register should not be written to until this bit is cleared. In the half-duplex mode, when this bit is set (and TFCE is set), then backpressure is asserted by the MAC. During backpressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. When the MAC is configured for the full-duplex mode, the BPA is automatically disabled. (R/WS/SC)(FCB)/(R/W)(BPA(backpressure activate))

Register 10.20: EMACDEBUG_REG (0x1024)

(reserved)	MTLTSFFS	MTLTFNES	(reserved)	MTLTFWCS	MTLTFRCS	MACTP	MACTFCS	MACTPES	(reserved)	MTLRFFLS	(reserved)	MTLRFRCS	MTLRFWCAS	(reserved)	MACRFFCS	MACRPES	
31 26	25	24	23	22	21 20	19	18 17	16	15 10	9 8	7	6 5	4	3	2 1	0	
0 0 0 0 0 0	0	0	0	0	0x0	0	0x0	0	0 0 0 0 0 0	0x0	0	0x0	0	0	0x0	0	Reset

MTLTSFFS When high, this bit indicates that the MTL TxStatus FIFO is full. Therefore, the MTL cannot accept any more frames for transmission. (RO)

MTLTFNES When high, this bit indicates that the MTL Tx FIFO is not empty and some data is left for transmission. (RO)

MTLTFWCS When high, this bit indicates that the MTL Tx FIFO Write Controller is active and is transferring data to the Tx FIFO. (RO)

MTLTFRCS This field indicates the state of the Tx FIFO Read Controller: (RO)

- 2'b00: IDLE state.
- 2'b01: READ state (transferring data to the MAC transmitter).
- 2'b10: Waiting for TxStatus from the MAC transmitter.
- 2'b11: Writing the received TxStatus or flushing the Tx FIFO.

MACTP When high, this bit indicates that the MAC transmitter is in the Pause condition (in the full-duplex-mode) and hence does not schedule any frame for transmission. (RO)

MACTFCS This field indicates the state of the MAC Transmit Frame Controller module: (RO)

- 2'b00: IDLE state.
- 2'b01: Waiting for status of previous frame or IFG or backoff period to be over.
- 2'b10: Generating and transmitting a Pause frame (in the full-duplex mode).
- 2'b11: Transferring input frame for transmission.

MACTPES When high, this bit indicates that the MAC MII transmit protocol engine is actively transmitting data and is not in the IDLE state. (RO)

MTLRFFLS This field gives the status of the fill-level of the Rx FIFO: (RO)

- 2'b00: Rx FIFO Empty.
- 2'b01: Rx FIFO fill-level below flow-control deactivate threshold.
- 2'b10: Rx FIFO fill-level above flow-control activate threshold.
- 2'b11: Rx FIFO Full.

Continued on the next page...

Register 10.20: EMACDEBUG_REG (0x1024)

Continued from the previous page ...

MTLRFRCS This field gives the state of the Rx FIFO read Controller: (RO)

2'b00: IDLE state.

2'b01: Reading frame data.

2'b10: Reading frame status (or timestamp).

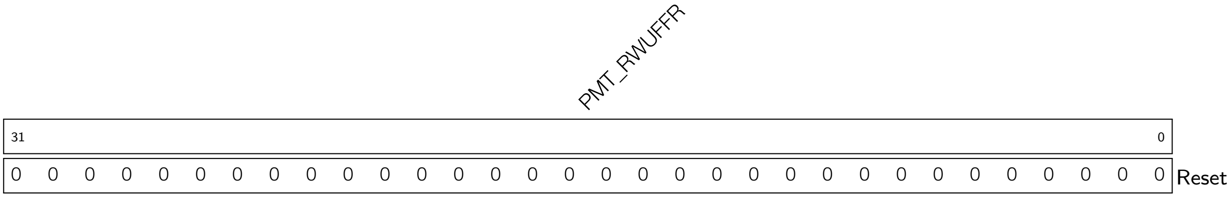
2'b11: Flushing the frame data and status.

MTLRFWCAS When high, this bit indicates that the MTL Rx FIFO Write Controller is active and is transferring a received frame to the FIFO. (RO)

MACRFFCS When high, this field indicates the active state of the FIFO Read and Write controllers of the MAC Receive Frame Controller Module. MACRFFCS[1] represents the status of FIFO Read controller. MACRFFCS[0] represents the status of small FIFO Write controller. (RO)

MACRPES When high, this bit indicates that the MAC MII receive protocol engine is actively receiving data and not in IDLE state. (RO)

Register 10.21: PMT_RWUFFR_REG (0x1028)



WKUPPKTFILTER The MSB (31st bit) must be zero. Bit j[30:0] is the byte mask. If Bit 1/2/3/4 (byte number) of the byte mask is set, the CRC block processes the filter 0/1/2/3 Offset + j of the incoming packet (RWKPTR is 0/1/2/3). (R/W)

- RWKPTR is 0:Filter 0 Byte Mask;
- RWKPTR is 1: Filter 1 Byte Mask;
- RWKPTR is 2: Filter 2 Byte Mask;
- RWKPTR is 3: Filter 3 Byte Mask;
- RWKPTR is 4: Bit 3/11/19/27 specifies the address type, defining the destination address type of the pattern. When the bit is set, the pattern applies to only multicast packets; when the bit is reset, the pattern applies only to unicast packet for filter 0/1/2/3. Bit 0/8/16/24 is the enable bit for filter 0/1/2/3;
- RWKPTR is 5: This filter 0/1/2/3 offset register defines the offset (within the packet) from which the filter 0/1/2/3 examines the packets;
- RWKPTR is 6: This filter 0 (bit[15:0])/1 (bit[31:16]) CRC16 register contains the CRC16 value calculated from the pattern and also the byte mask programmed to the wake-up filter register block; The polynomial:
$$G(x) = x^{16} + x^{15} + x^2 + 1.$$
- RWKPTR is 7: This filter 2 bit[15:0])/3(bit[31:16]) CRC16 register contains the CRC16 value calculated from the pattern and also the byte mask programmed to the wake-up filter register block. The polynomial:
$$G(x) = x^{16} + x^{15} + x^2 + 1.$$

Register 10.22: PMT_CSR_REG (0x102C)

RWKFILTRST	(reserved)	RWKPTR	(reserved)	GLBLUCAST	(reserved)	RWKPRCVD	MGKPRCVD	(reserved)	RWKPKTEN	MGKPKTEN	PWRDWN	
31	30 29	28 24	23 10	9	8 7	6	5	4 3	2	1	0	
0	0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	00	0 0	0	0	0 0	0	0	0	Reset

RWKFILTRST When this bit is set, it resets the remote RWKPTR register to 3'b000. (R/WS/SC)

RWKPTR The maximum value of the pointer is 7 ,the detail information ,please refer to PMT_RWUFFR. (RO)

GLBLUCAST When set, enables any unicast packet filtered by the MAC (DAFilter) address recognition to be a remote wake-up frame. (R/W)

RWKPRCVD When set, this bit indicates the power management event is generated because of the reception of a remote wake-up frame. This bit is cleared by a Read into this register. (R/SS/RC)

MGKPRCVD When set, this bit indicates that the power management event is generated because of the reception of a magic packet. This bit is cleared by a Read into this register. (R/SS/RC)

RWKPKTEN hen set, enables generation of a power management event because of remote wake-up frame reception. (R/W)

MGKPKTEN When set, enables generation of a power management event because of magic packet reception. (R/W)

PWRDWN hen set, the MAC receiver drops all received frames until it receives the expected magic packet or remote wake-up frame. This bit must only be set when MGKPKTEN, GLBLUCAST, or RWKPKTEN bit is set high. (R/WS/SC)

Register 10.23: EMACLPI_CSR_REG (0x1030)

(reserved)	LPITXA	(reserved)	PLS	LPIEN	(reserved)	RLPIST	TLPIST	(reserved)	RLPIEX	RLPIEN	TLPIEX	TLPIEN	
31 20	19	18	17	16	15 10	9	8	7 4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0 0 0 0 0	0	0	0	0	0	0	0	Reset

LPITXA This bit controls the behavior of the MAC when it is entering or coming out of the LPI mode on the transmit side.If the LPITXA and LPIEN bits are set to 1, the MAC enters the LPI mode only after all outstanding frames and pending frames have been transmitted. The MAC comes out of the LPI mode when the application sends any frame.When this bit is 0, the LPIEN bit directly controls behavior of the MAC when it is entering or coming out of the LPI mode. (R/W)

PLS This bit indicates the link status of the PHY. When set, the link is considered to be okay (up) and when reset, the link is considered to be down. (R/W)

LPIEN When set, this bit instructs the MAC Transmitter to enter the LPI state. When reset, this bit instructs the MAC to exit the LPI state and resume normal transmission.This bit is cleared when the LPITXA bit is set and the MAC exits the LPI state because of the arrival of a new packet for transmission. (R/W/SC)

RLPIST When set, this bit indicates that the MAC is receiving the LPI pattern on the MII interface. (R/W)

TLPIST When set, this bit indicates that the MAC is receiving the LPI pattern on the MII interface. (R/W)

RLPIEX When set, this bit indicates that the MAC Receiver has stopped receiving the LPI pattern on the MII interface, exited the LPI state, and resumed the normal reception. This bit is cleared by a read into this register. (R/SS/RC)

RLPIEN When set, this bit indicates that the MAC Receiver has received an LPI pattern and entered the LPI state. This bit is cleared by a read into this register. (R/SS/RC)

TLPIEX When set, this bit indicates that the MAC transmitter has exited the LPI state after the user has cleared the LPIEN bit and the LPI_TW_Timer has expired.This bit is cleared by a read into this register. (R/SS/RC)

TLPIEN When set, this bit indicates that the MAC Transmitter has entered the LPI state because of the setting of the LPIEN bit. This bit is cleared by a read into this register. (R/SS/RC)

Register 10.24: EMACLPITIMERSCONTROL_REG (0x1034)

(reserved)	LPI_LS_TIMER	LPI_TW_TIMER	
31 26	25 16	15 0	
0 0 0 0 0 0	0x3E8	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

LPI_LS_TIMER This field specifies the minimum time (in milliseconds) for which the link status from the PHY should be up (OKAY) before the LPI pattern can be transmitted to the PHY. The MAC does not transmit the LPI pattern even when the LPIEN bit is set unless the LPI_LS_Timer reaches the programmed terminal count. The default value of the LPI_LS_Timer is 1000 (1 sec) as defined in the IEEE standard.(R/W)

LPI_TW_TIMER This field specifies the minimum time (in microseconds) for which the MAC waits after it stops transmitting the LPI pattern to the PHY and before it resumes the normal transmission. The TLPIEX status bit is set after the expiry of this timer.(R/W)

Register 10.25: EMACINTS_REG (0x1038)

(reserved)	LPIINTS	(reserved)	(reserved)	PMTINTS	(reserved)	
31 11	10	9	8 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0	0	0 0 0	Reset

LPIINTS When the Energy Efficient Ethernet feature is enabled, this bit is set for any LPI state entry or exit in the MAC Transmitter or Receiver. This bit is cleared on reading Bit[0] of Register (LPI Control and Status Register). (RO)

PMTINTS This bit is set when a magic packet or remote wake-up frame is received in the power-down mode (see Bit[5] and Bit[6] in the PMT Control and Status Register). This bit is cleared when both Bits[6:5] are cleared because of a read operation to the PMT Control and Status register. This bit is valid only when you select the optional PMT module during core configuration. (RO)

Register 10.26: EMACINTMASK_REG (0x103C)

(reserved)	LPIINTMASK	(reserved)	(reserved)	PMTINTMASK	(reserved)	
31 11	10	9	8 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0	0	0 0 0	Reset

LPIINTMASK When set, this bit disables the assertion of the interrupt signal because of the setting of the LPI Interrupt Status bit in Register (Interrupt Status Register). (R/W)

PMTINTMASK When set, this bit disables the assertion of the interrupt signal because of the setting of PMT Interrupt Status bit in Register (Interrupt Status Register). (R/W)

Register 10.27: EMACADDR0HIGH_REG (0x1040)

ADDRESS_ENABLE0	(reserved)	MAC_ADDRESS0_HI	
31	30 16	15 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE0 This bit is always set to 1. (RO)

MAC_ADDRESS0_HI This field contains the upper 16 bits (47:32) of the first 6-byte MAC address. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (Pause) Frames. (R/W)

Register 10.28: EMACADDR0LOW_REG (0x1044)

31 0	
0x0FFFFFFFF	Reset

EMACADDR0LOW_REG This field contains the lower 32 bits of the first 6-byte MAC address. This is used by the MAC for filtering the received frames and inserting the MAC address in the Transmit Flow Control (Pause) Frames. (R/W)

Register 10.29: EMACADDR1HIGH_REG (0x1048)

ADDRESS_ENABLE1	SOURCE_ADDRESS	MASK_BYTE_CONTROL	(reserved)	MAC_ADDRESS1_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE1 When this bit is set, the address filter module uses the second MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS When this bit is set, the EMACADDR1[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR1[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL These bits are mask control bits for comparison of each of the EMACADDR1 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR1 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR1 High [15:8].
- Bit[28]: EMACADDR1 High [7:0].
- Bit[27]: EMACADDR1 Low [31:24].
- Bit[24]: EMACADDR1 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS1_HI This field contains the upper 16 bits, Bits[47:32] of the second 6-byte MAC address. (R/W)

Register 10.30: EMACADDR1LOW_REG (0x104C)

31 0	
0x0FFFFFFFF	Reset

EMACADDR1LOW_REG This field contains the lower 32 bits of the second 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.31: EMACADDR2HIGH_REG (0x1050)

ADDRESS_ENABLE2	SOURCE_ADDRESS2	MASK_BYTE_CONTROL2	(reserved)	MAC_ADDRESS2_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE2 When this bit is set, the address filter module uses the third MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS2 When this bit is set, the EMACADDR2[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR2[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL2 These bits are mask control bits for comparison of each of the EMACADDR2 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR2 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR2 High [15:8].
- Bit[28]: EMACADDR2 High [7:0].
- Bit[27]: EMACADDR2 Low [31:24].
- Bit[24]: EMACADDR2 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS2_HI This field contains the upper 16 bits, Bits[47:32] of the third 6-byte MAC address. (R/W)

Register 10.32: EMACADDR2LOW_REG (0x1054)

31 0	
0x0FFFFFFFF	Reset

EMACADDR2LOW_REG This field contains the lower 32 bits of the third 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.33: EMACADDR3HIGH_REG (0x1058)

ADDRESS_ENABLE3	SOURCE_ADDRESS3	MASK_BYTE_CONTROL3	(reserved)	MAC_ADDRESS3_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE3 When this bit is set, the address filter module uses the fourth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS3 When this bit is set, the EMACADDR3[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR3[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL3 These bits are mask control bits for comparison of each of the EMACADDR3 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR3 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR3 High [15:8].
- Bit[28]: EMACADDR3 High [7:0].
- Bit[27]: EMACADDR3 Low [31:24].
- Bit[24]: EMACADDR3 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS3_HI This field contains the upper 16 bits, Bits[47:32] of the fourth 6-byte MAC address. (R/W)

Register 10.34: EMACADDR3LOW_REG (0x105C)

31 0	
0x0FFFFFFFF	Reset

EMACADDR3LOW_REG This field contains the lower 32 bits of the fourth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.35: EMACADDR4HIGH_REG (0x1060)

ADDRESS_ENABLE4	SOURCE_ADDRESS4	MASK_BYTE_CONTROL4	(reserved)	MAC_ADDRESS4_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE4 When this bit is set, the address filter module uses the fifth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS4 When this bit is set, the EMACADDR4[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR4[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL4 These bits are mask control bits for comparison of each of the EMACADDR4 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR4 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR4 High [15:8].
- Bit[28]: EMACADDR4 High [7:0].
- Bit[27]: EMACADDR4 Low [31:24].
- Bit[24]: EMACADDR4 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS4_HI This field contains the upper 16 bits, Bits[47:32] of the fifth 6-byte MAC address. (R/W)

Register 10.36: EMACADDR4LOW_REG (0x1064)

31 0	
0x0FFFFFFFF	Reset

EMACADDR4LOW_REG This field contains the lower 32 bits of the fifth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.37: EMACADDR5HIGH_REG (0x1068)

ADDRESS_ENABLE5	SOURCE_ADDRESS5	MASK_BYTE_CONTROL5	(reserved)	MAC_ADDRESS5_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE5 When this bit is set, the address filter module uses the sixth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS5 When this bit is set, the EMACADDR5[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR5[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL5 These bits are mask control bits for comparison of each of the EMACADDR5 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR5 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR5 High [15:8].
- Bit[28]: EMACADDR5 High [7:0].
- Bit[27]: EMACADDR5 Low [31:24].
- Bit[24]: EMACADDR5 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS5_HI This field contains the upper 16 bits, Bits[47:32] of the sixth 6-byte MAC address. (R/W)

Register 10.38: EMACADDR5LOW_REG (0x106C)

31 0	
0x0FFFFFFFF	Reset

EMACADDR5LOW_REG This field contains the lower 32 bits of the sixth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.39: EMACADDR6HIGH_REG (0x1070)

ADDRESS_ENABLE6	SOURCE_ADDRESS6	MASK_BYTE_CONTROL6	(reserved)	MAC_ADDRESS6_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE6 When this bit is set, the address filter module uses the seventh MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS6 When this bit is set, the EMACADDR6[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR6[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL6 These bits are mask control bits for comparison of each of the EMACADDR6 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR6 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR6 High [15:8].
- Bit[28]: EMACADDR6 High [7:0].
- Bit[27]: EMACADDR6 Low [31:24].
- Bit[24]: EMACADDR6 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS6_HI This field contains the upper 16 bits, Bits[47:32] of the seventh 6-byte MAC address. (R/W)

Register 10.40: EMACADDR6LOW_REG (0x1074)

31 0	
0x0FFFFFFFF	Reset

EMACADDR6LOW_REG This field contains the lower 32 bits of the seventh 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.41: EMACADDR7HIGH_REG (0x1078)

ADDRESS_ENABLE7	SOURCE_ADDRESS7	MASK_BYTE_CONTROL7	(reserved)	MAC_ADDRESS7_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE7 When this bit is set, the address filter module uses the eighth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering. (R/W)

SOURCE_ADDRESS7 When this bit is set, the EMACADDR7[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the EMACADDR7[47:0] is used to compare with the DA fields of the received frame. (R/W)

MASK_BYTE_CONTROL7 These bits are mask control bits for comparison of each of the EMACADDR7 bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of EMACADDR7 registers. Each bit controls the masking of the bytes as follows:

- Bit[29]: EMACADDR7 High [15:8].
- Bit[28]: EMACADDR7 High [7:0].
- Bit[27]: EMACADDR7 Low [31:24].
- Bit[24]: EMACADDR7 Low [7:0].

You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address. (R/W)

MAC_ADDRESS7_HI This field contains the upper 16 bits, Bits[47:32] of the eighth 6-byte MAC address. (R/W)

Register 10.42: EMACADDR7LOW_REG (0x107C)

31 0	
0x0FFFFFFFF	Reset

EMACADDR7LOW_REG This field contains the lower 32 bits of the eighth 6-byte MAC address. The content of this field is undefined, so the register needs to be configured after the initialization process. (R/W)

Register 10.43: EMACCSTATUS_REG (0x10D8)

(reserved)	SMIDRXS	(reserved)	(JABBER_TIMEOUT)	(reserved)	LINK_SPEED	LINK_MODE	
31 17	16	15 5	4	3	2 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

JABBER_TIMEOUT This bit indicates whether there is jabber timeout error (1'b1) in the received frame. (RO)

LINK_SPEED This bit indicates the current speed of the link: (RO)

- 2'b00: 2.5 MHz.
- 2'b01: 25 MHz.
- 2'b10: 125 MHz.

LINK_MODE This bit indicates the current mode of operation of the link: (RO)

- 1'b0: Half-duplex mode.
- 1'b1: Full-duplex mode.

Register 10.44: EMACWDOGTO_REG (0x10DC)

(reserved)	PWDOGEN	(reserved)	WDOGTO	
31 17	16	15 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0	0x0000	Reset

PWDOGEN When this bit is set and Bit[23] (WD) of EMACCONFIG_REG is reset, the WTO field (Bits[13:0]) is used as watchdog timeout for a received frame. When this bit is cleared, the watchdog timeout for a received frame is controlled by the setting of Bit[23] (WD) and Bit[20] (JE) in EMACCONFIG_REG. (R/W)

WDOGTO When Bit[16] (PWE) is set and Bit[23] (WD) of EMACCONFIG_REG is reset, this field is used as watchdog timeout for a received frame. If the length of a received frame exceeds the value of this field, such frame is terminated and declared as an error frame. (R/W)

Register 10.45: EMAC_EX_CLKOUT_CONF_REG (0x0000)

(reserved)	EMAC_CLK_OUT_H_DIV_NUM	EMAC_CLK_OUT_DIV_NUM	
31 8	7 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x02	0x04	Reset

EMAC_CLK_OUT_H_DIV_NUM RMII CLK using internal PLLA CLK, the half divider number, when using RMII PHY. (R/W)

EMAC_CLK_OUT_DIV_NUM RMII CLK using internal PLLA CLK, the whole divider number, when using RMII PHY. (R/W)

Register 10.46: EMAC_EX_OSCCLK_CONF_REG (0x0004)

(reserved)	EMAC_OSC_CLK_SEL	EMAC_OSC_H_DIV_NUM_100M	EMAC_OSC_DIV_NUM_100M	EMAC_OSC_H_DIV_NUM_10M	EMAC_OSC_DIV_NUM_10M	
31 25	24	23 18	17 12	11 6	5 0	
0 0 0 0 0 0 0	0	0	1	9	19	Reset

EMAC_OSC_CLK_SEL Ethernet work using external PHY output clock or not for RMII CLK, when using RMII PHY. When this bit is set to 1, external PHY CLK is used. When this bit is set to 0, PLLA CLK is used. (R/W)

EMAC_OSC_H_DIV_NUM_100M RMII/MII half-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 100M. (R/W)

EMAC_OSC_DIV_NUM_100M RMII/MII whole-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 100M. (R/W)

EMAC_OSC_H_DIV_NUM_10M RMII/MII half-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 10M. (R/W)

EMAC_OSC_DIV_NUM_10M RMII/MII whole-integer divider, when register EMAC_EX_CLKOUT_CONF clock divider's speed is 10M. (R/W)

Register 10.47: EMAC_EX_CLK_CTRL_REG (0x0008)

(reserved)	(reserved)	EMAC_MII_CLK_RX_EN	EMAC_MII_CLK_TX_EN	(reserved)	EMAC_INT_OSC_EN	EMAC_EXT_OSC_EN	
31 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

EMAC_MII_CLK_RX_EN Enable Ethernet RX CLK. (R/W)

EMAC_MII_CLK_TX_EN Enable Ethernet TX CLK. (R/W)

EMAC_INT_OSC_EN Using internal PLLA CLK in RMII PHY mode. (R/W)

EMAC_EXT_OSC_EN Using external PLLA CLK in RMII PHY mode. (R/W)

Register 10.48: EMAC_EX_PHYINF_CONF_REG (0x000C)

(reserved)	EMAC_PHY_INTF_SEL	(reserved)	
31 16	15 13	25 13	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EMAC_PHY_INTF_SEL The PHY interface selected. 0x0: PHY MII, 0x4: PHY RMII. (R/W)

Register 10.49: EMAC_PD_SEL_REG (0x0010)

(reserved)	EMAC_RAM_PD_EN	
31 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

EMAC_RAM_PD_EN Ethernet RAM power-down enable signal. Bit[0]: TX SRAM; Bit[1]: RX SRAM. Setting the bit to 1 powers down the RAM. (R/W)

11. I^{2}C Controller

11.1 Overview

An I^{2}C (Inter-Integrated Circuit) bus can be used for communication with several external devices connected to the same bus as ESP32. The ESP32 has dedicated hardware to communicate with peripherals on the I^{2}C bus.

11.2 Features

The I^{2}C controller has the following features:

- Supports both master mode and slave mode
- Supports multi-master and multi-slave communication
- Supports standard mode (100 kbit/s)
- Supports fast mode (400 kbit/s)
- Supports 7-bit addressing and 10-bit addressing
- Supports continuous data transmission with disabled Serial Clock Line (SCL)
- Supports programmable digital noise filter

11.3 Functional Description

11.3.1 Introduction

I^{2}C is a two-wire bus, consisting of an SDA and an SCL line. These lines are configured to open the drain output. The lines are shared by two or more devices: usually one or more masters and one or more slaves.

Communication starts when a master sends out a start condition: it will pull the SDA line low, and will then pull the SCL line high. It will send out nine clock pulses over the SCL line. The first eight pulses are used to shift out a byte consisting of a 7-bit address and a read/write bit. If a slave with this address is active on the bus, the slave can answer by pulling the SDA low on the ninth clock pulse. The master can then send out more 9-bit clock pulse clusters and, depending on the read/write bit sent, the device or the master will shift out data on the SDA line, with the other side acknowledging the transfer by pulling the SDA low on the ninth clock pulse. During data transfer, the SDA line changes only when the SCL line is low. When the master has finished the communication, it will send a stop condition on the bus by raising SDA, while SCL will already be high.

The ESP32 I^{2}C peripheral can handle the I^{2}C protocol, freeing up the processor cores for other tasks.

11.3.2 Architecture

An I²C controller can operate either in master mode or slave mode. The I2C_MS_MODE register is used to select the mode. Figure 48 shows the I²C Master architecture, while Figure 49 shows the I²C Slave architecture.

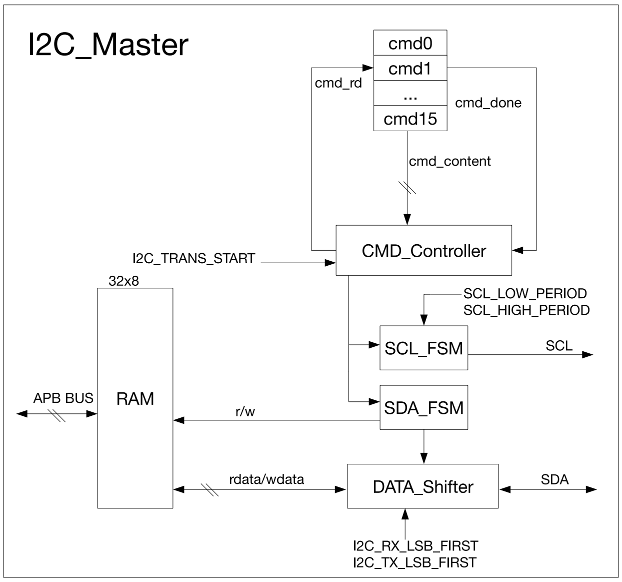


Figure 48: I²C Master Architecture

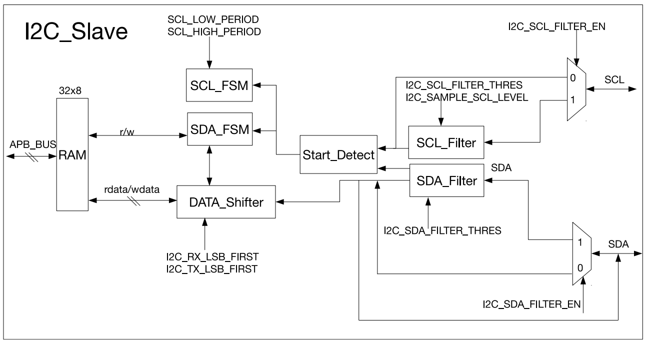


Figure 49: I²C Slave Architecture

The I²C controller contains the following units:

- RAM, the size of which is 32 x 8 bits, and it is directly mapped onto the address space of the CPU cores, starting at address REG_I2C_BASE+0x100. Each byte of I²C data is stored in a 32-bit word of memory (so, the first byte is at +0x100, the second byte at +0x104, the third byte at +0x108, etc.) Users need to set register I2C_NONFIFO_EN.
- A CMD_Controller and 16 command registers (cmd0 ~ cmd15), which are used by the I²C Master to control data transmission. One command at a time is executed by the I²C controller.
- SCL_FSM: A state machine that controls the SCL clock. The I2C_SCL_HIGH_PERIOD_REG and I2C_SCL_LOW_PERIOD_REG registers are used to configure the frequency and duty cycle of the signal on the SCL line.
- SDA_FSM: A state machine that controls the SDA data line.
- DATA_Shifter which converts the byte data to an outgoing bitstream, or converts an incoming bitstream to byte data. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used for configuring whether the LSB or MSB is stored or transmitted first.

- SCL_Filter and SDA_Filter: Input noise filter for the I2C_Slave. The filter can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN. The filter can remove line glitches with pulse width less than I2C_SCL_FILTER_THRES and I2C_SDA_FILTER_THRES ABP clock cycles.

11.3.3 I²C Bus Timing

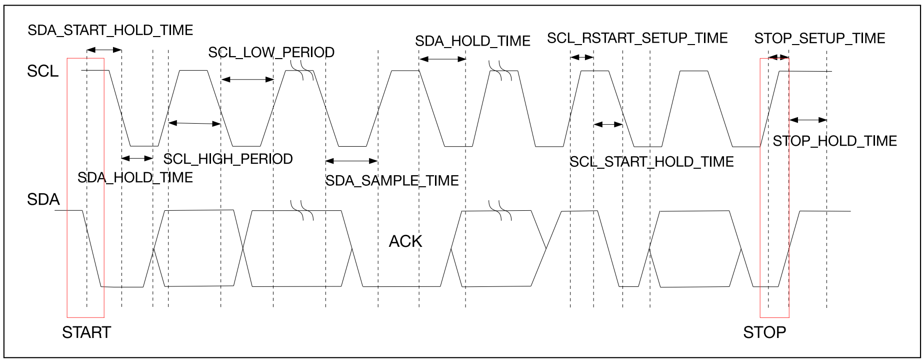


Figure 50: I²C Sequence Chart

Figure 50 is an I²C sequence chart. When the I²C controller works in master mode, SCL is an output signal. In contrast, when the I²C controller works in slave mode, the SCL becomes an input signal. The values assigned to I2C_SDA_HOLD_REG and I2C_SDA_SAMPLE_REG are still valid in slave mode. Users need to configure the values of I2C_SDA_HOLD_TIME and I2C_SDA_SAMPLE_TIME, according to the host characteristics, for the I²C slave to receive data properly. Table 54 shows available settings of SCL low and high level cycles when SCL is configured to direct output mode. The settings determine the SCL output frequency f_{scl}.

Table 54: SCL Frequency Configuration

I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	SCL_Low_Level_Cycles	SCL_High_Level_Cycles
0	Don't care	I2C_SCL_LOW_PERIOD+1	I2C_SCL_HIGH_PERIOD+7
1	[0,2]		I2C_SCL_HIGH_PERIOD+8
	[3,7]		I2C_SCL_HIGH_PERIOD+6+I2C_SCL_FILTER_THRES

$$f_{\text{scl}} = \frac{80\text{ MHz}}{\text{SCL\_Low\_Level\_Cycles} + \text{SCL\_High\_Level\_Cycles}}$$

According to the I²C protocol, each transmission of data begins with a START condition and ends with a STOP condition. Data is transmitted by one byte at a time, and each byte has an ACK bit. The receiver informs the transmitter to continue transmission by pulling down SDA, which indicates an ACK. The receiver can also indicate it wants to stop further transmission by pulling up the SDA line, thereby not indicating an ACK.

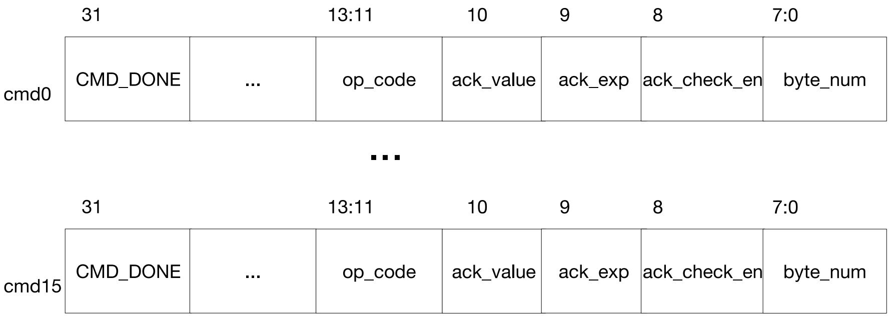
Figure 50 also shows the registers that can configure the START bit, STOP bit, SDA hold time, and SDA sample time.

Notice: If the I²C pads are configured in open-drain mode, it will take longer for the signal lines to transition from a low level to a high level. The transition duration is determined together by the pull-up resistor and capacitor. The output frequency of SCL is relatively low in open-drain mode.

11.3.4 I²C cmd Structure

The Command register is active only in I²C master mode, with its internal structure shown in Figure 51.

CMD_DONE: The CMD_DONE bit of every command can be read by software to tell if the command has been handled by hardware.



	31		13:11	10	9	8	7:0
cmd0	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

...

	31		13:11	10	9	8	7:0
cmd15	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

Figure 51: Structure of The I^{2}C Command Register

op_code: op_code is used to indicate the command. The I^{2}C controller supports four commands:

- RSTART: op_code = 0 is the RSTART command to control the transmission of a START or RESTART I^{2}C condition.
- WRITE: op_code = 1 is the WRITE command for the I^{2}C Master to transmit data.
- READ: op_code = 2 is the READ command for the I^{2}C Master to receive data.
- STOP: op_code = 3 is the STOP command to control the transmission of a STOP I^{2}C condition.
- END: op_code = 4 is the END command for continuous data transmission. When the END command is given, SCL is temporarily disabled to allow software to reload the command and data registers for subsequent events before resuming. Transmission will then continue seamlessly.

A complete data transmission process begins with an RSTART command, and ends with a STOP command.

ack_value: When receiving data, this bit is used to indicate whether the receiver will send an ACK after this byte has been received.

ack_exp: This bit is to set an expected ACK value for the transmitter.

ack_check_en: When transmitting a byte, this bit enables checking the ACK value received against the ack_exp value. Checking is enabled by 1, while 0 disables it.

byte_num: This register specifies the length of data (in bytes) to be read or written. The maximum length is 255, while the minimum is 1. When the op_code is RSTART, STOP or END, this value is meaningless.

11.3.5 I^{2}C Master Writes to Slave

In all subsequent figures that illustrate I^{2}C transactions and behavior, both the I^{2}C Master and Slave devices are assumed to be ESP32 I^{2}C peripheral controllers for ease of demonstration.

Figure 52 shows the I^{2}C Master writing N bytes of data to an I^{2}C Slave. According to the I^{2}C protocol, the first byte is the Slave address. As shown in the diagram, the first byte of the RAM unit has been populated with the Slave's 7-bit address plus the 1-bit read/write flag. In this case, the flag is zero, indicating a write operation. The rest of the RAM unit holds N bytes of data ready for transmission. The cmd unit has been populated with the sequence of commands for the operation.

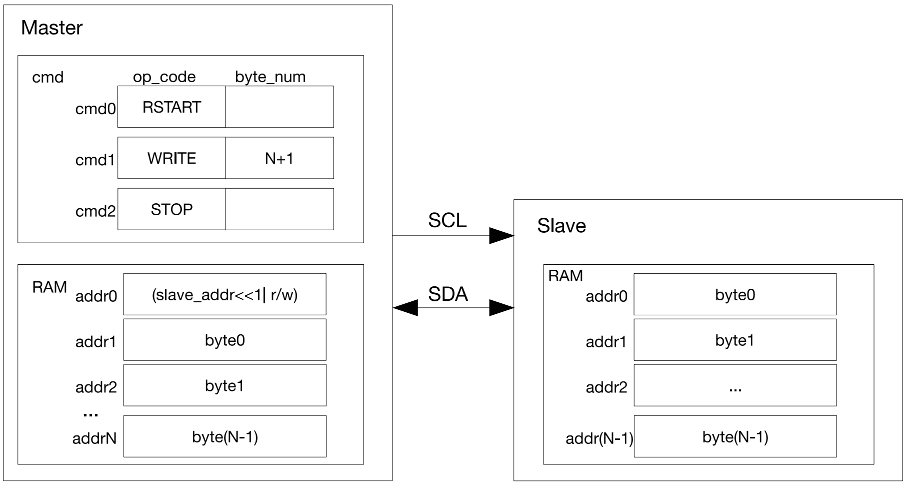


Figure 52: I²C Master Writes to Slave with 7-bit Address

For the I²C master to begin an operation, the bus must not be busy, i.e. the SCL line must not be pulled low by another device on the I²C bus. The I²C operation can only begin when the SCL line is released (made high) to indicate that the I²C bus is free. After the cmd unit and data are prepared, I2C_TRANS_START bit in I2C_CTR_REG must be set to begin the configured I²C Master operation. The I²C Master then initiates a START condition on the bus and progresses to the WRITE command which will fetch N+1 bytes from RAM and send them to the Slave. The first of these bytes is the address byte.

When the transmitted data size exceeds I2C_NONFIFO_TX_THRES, an I2C_TX_SEND_EMPTY_INT interrupt will be generated. After detecting the interrupt, software can read TXFIFO_END_ADDR in register RXFIFO_ST_REG, get the last address of the data in the RAM and refresh the old data in the RAM. TXFIFO_END_ADDR will be refreshed each time interrupt I2C_TX_SEND_EMPTY_INT or I2C_TRANS_COMPLETE_INT occurs.

When ack_check_en is set to 1, the Master will check the ACK value each time it sends a data byte. If the ACK value received does not match ack_exp (the expected ACK value) in the WRITE command, then the Master will generate an I2C_ACK_ERR_INT interrupt and stop the transmission.

During transmission, when the SCL is high, if the input value and output value of SDA do not match, then the Master will generate an I2C_ARBITRATION_LOST_INT interrupt. When the transmission is finished, the Master will generate an I2C_TRANS_COMPLETE_INT interrupt.

After detecting the START bit sent from the Master, the Slave will start receiving the address and comparing it to its own. If the address does not match I2C_SLAVE_ADDR, then the Slave will ignore the rest of the transmission. If they do match, the Slave will store the rest of the data into RAM in the receiving order. When the data size exceeds I2C_NONFIFO_RX_THRES, an I2C_RX_REC_FULL_INT interrupt is generated. After detecting the interrupt, software will get the starting and ending addresses in the RAM by reading RXFIFO_START_ADDR and RXFIFO_END_ADDR bits in register RXFIFO_ST_REG, and fetch the data for further processing. Register RXFIFO_START_ADDR is refreshed only once during each transmission, while RXFIFO_END_ADDR gets refreshed every time when either I2C_RX_REC_FULL_INT or I2C_TRANS_COMPLETE_INT interrupt is generated.

When the END command is not used, the I²C master can transmit up to (14*255-1) bytes of valid data, and the cmd unit is populated with RSTART + 14 WRITE + 1 STOP.

There are several special cases to be noted:

- If the Master fails to send a STOP bit, because the SDA is pulled low by other devices, then the Master needs to be reset.

- If the Master fails to send a START bit, because the SDA or SCL is pulled low by other devices, then the Master needs to be reset. It is recommended that the software uses a timeout period to implement the reset.
- If the SDA is pulled low by the Slave during transmission, the Master can simply release it by sending it nine SCL clock signals at the most.

It is important to note that the behaviour of another I^{2}C master or slave device on the bus may not always be similar to that of the ESP32 I^{2}C peripheral in the master- or slave-mode operation described above. Please consult the datasheets of the respective I^{2}C devices to ensure proper operation under all bus conditions.

The ESP32 I^{2}C controller uses 7-bit addressing by default. However, 10-bit addressing can also be used. In the master, this is done by sending a second I^{2}C address byte after the first address byte. In the slave, the I2C_SLAVE_ADDR_10BIT_EN bit in I2C_SLAVE_ADDR_REG can be set to activate a 10-bit addressing mode. I2C_SLAVE_ADDR is used to configure the I^{2}C Slave address, as per usual. Figure 53 shows the equivalent of I^{2}C Master operation writing N-bytes of data to an I^{2}C Slave with a 10-bit address. Since 10-bit Slave addresses require an extra address byte, both the byte_num field of the WRITE command and the number of total bytes in RAM increase by one.

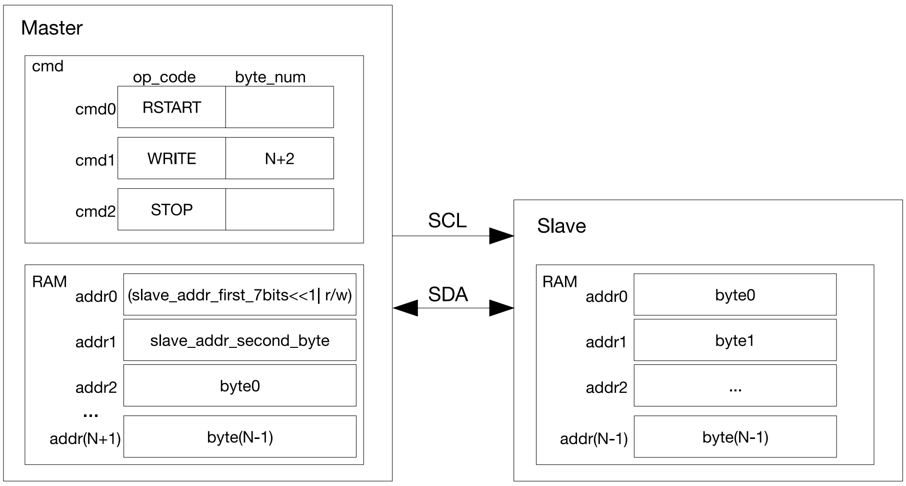


Figure 53: I^{2}C Master Writes to Slave with 10-bit Address

When the END command is not used, the I^{2}C master can transmit up to (14*255-2) bytes of valid data to Slave with 10-bit address.

One way many I^{2}C Slave devices are designed is by exposing a register block containing various settings. The I^{2}C Master can write one or more of these registers by sending the Slave a register address. The ESP32 I^{2}C Slave controller has hardware support for such a scheme.

Specifically, on the Slave, I2C_FIFO_ADDR_CFG_EN can be set so that the I^{2}C Master can write to a specified register address inside the I^{2}C Slave memory block. Figure 54 shows the Master writing N-bytes of data byte0 ~ byte(N-1) from the RAM unit to register address M (determined by addrM in RAM unit) with the Slave. In this mode, Slave can receive up to 32 bytes of valid data. When Master needs to transmit extra amount of data, segmented transmission can be enabled.

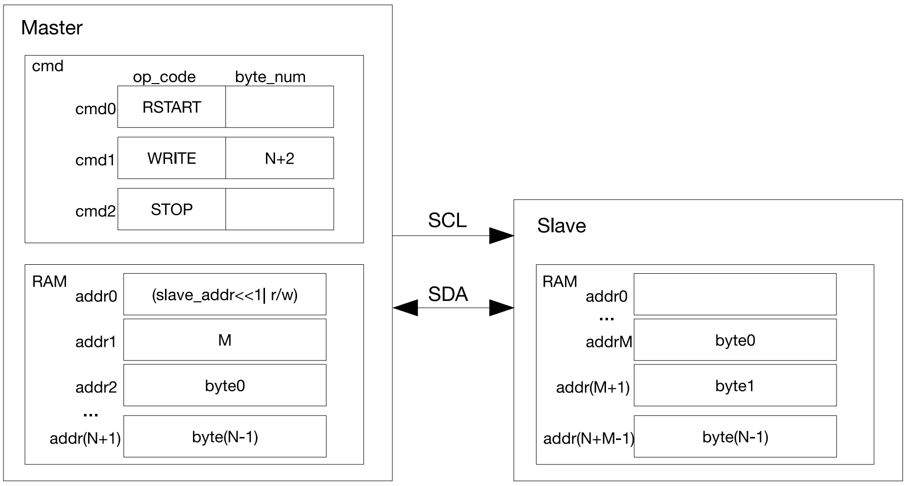


Figure 54: I²C Master Writes to addrM in RAM of Slave with 7-bit Address

If the data size exceeds the capacity of a 14-byte read/write cmd, the END command can be called to enable segmented transmission. Figure 55 shows the Master writing data to the Slave, in three segments. The first segment shows the configuration of the Master's commands and the preparation of data in the RAM unit. When the I2C_TRANS_START bit is enabled, the Master starts transmission. After executing the END command, the Master will turn off the SCL clock and pull the SCL low to reserve the bus and prevent any other device from transacting on the bus. The controller will generate an I2C_END_DETECT_INT interrupt to notify the software.

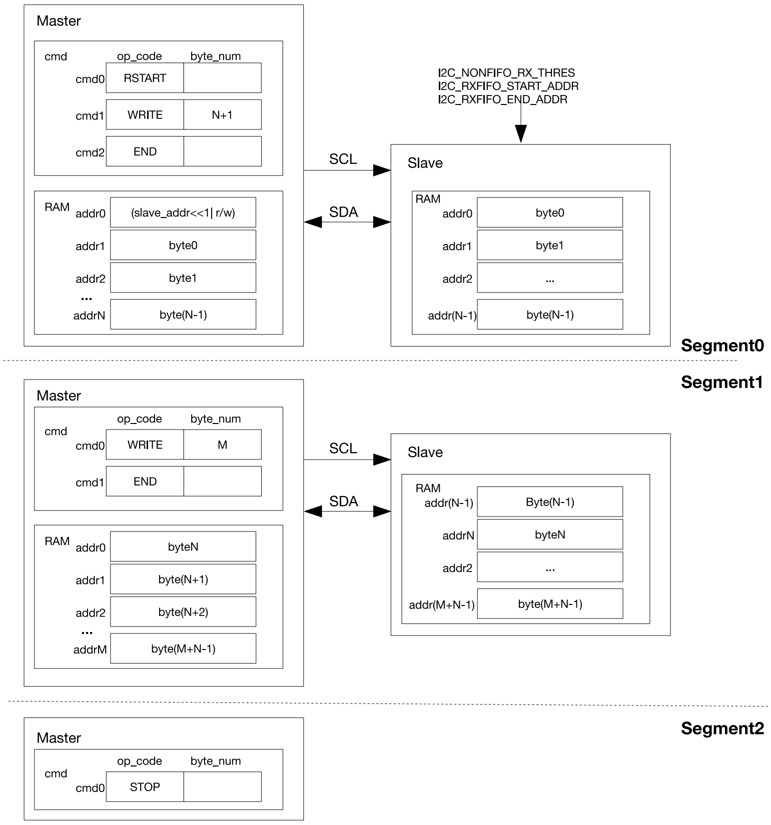


Figure 55: Master Writes to Slave with 7-bit Address in Three Segments

After detecting an I2C_END_DETECT_INT interrupt, the software can refresh the contents of the cmd and RAM blocks, as shown in the second segment. Subsequently, it should clear the I2C_END_DETECT_INT interrupt and resume the transaction by setting the I2C_TRANS_START bit. To stop the transaction, it should configure the cmd, as the third segment shows, and enable the I2C_TRANS_START bit to generate a STOP bit, after detecting the I2C_END_DETECT_INT interrupt.

Please note that the other masters on the bus will be starved of bus time between two segments. The bus is only released after a STOP signal is sent.

Note: When there are more than three segments, the address of an END command in the cmd should not be altered into another command by the next segment.

11.3.6 Master Reads from Slave

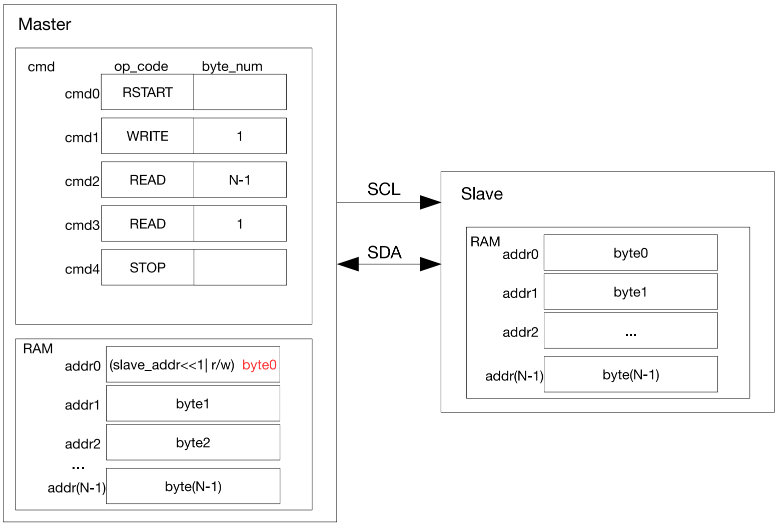


Figure 56: Master Reads from Slave with 7-bit Address

Figure 56 shows the Master reading N-bytes of data from an Slave with a 7-bit address. At first, the Master needs to send the address of the Slave, so cmd1 is a WRITE command. The byte that this command sends is the slave address plus the R/W flag, which in this case is 1 and, therefore, indicates that this is going to be a read operation. The Slave starts to send data to the Master if the addresses match. The Master will return ACK, according to the ack_value in the READ command, upon receiving every byte. As can be seen from Figure 56, READ is divided into two segments. The Master replies ACK to N-1 bytes in cmd2 and does not reply ACK to the single byte READ command in cmd3, i.e., the last transmitted data. Users can configure it as they wish.

When storing the received data, Master will start from the first address in RAM. Byte0 (Slave address + 1-bit R/W marker bit) will be overwritten.

When the END command is not used, the Master can transmit up to (13*255) bytes of valid data. The cmd unit is populated with RSTART + 1 WRITE + 13 READ + 1 STOP.

Figure 57 shows the Master reading data from a slave with a 10-bit address. This mode can be enabled by setting I2C_SLAVE_ADDR_10BIT_EN bit and preparing data to be sent in the slave RAM. In the Master, two bytes of RAM are used for a 10-bit address. Finally, the I2C _TRANS_START bit must be set to enable one transaction.

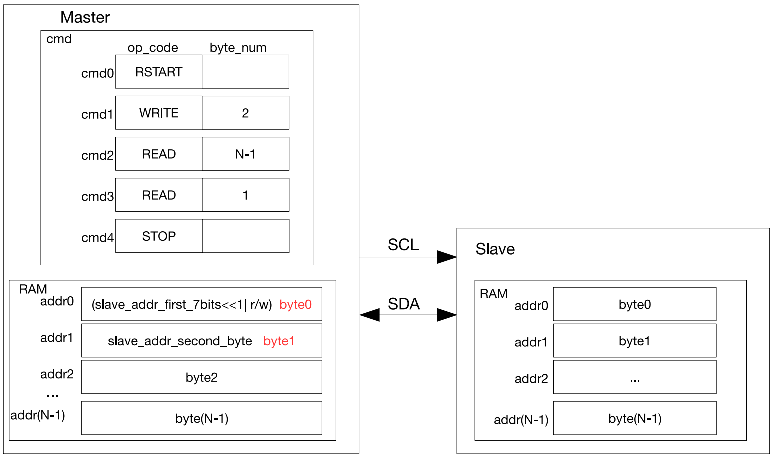


Figure 57: Master Reads from Slave with 10-bit Address

Figure 58 shows the Master reading data from a specified address in the Slave. This mode can be enabled by setting I2C_FIFO_ADDR_CFG_EN and preparing the data to be read by the master in the Slave RAM block. Subsequently, the address of the Slave and the address of the specified register (that is, M) have to be determined by the master. Finally, the I2C_TRANS_START bit must be set in the Master to initiate the read operation, following which the Slave will fetch N bytes of data from RAM and send them to the Master.

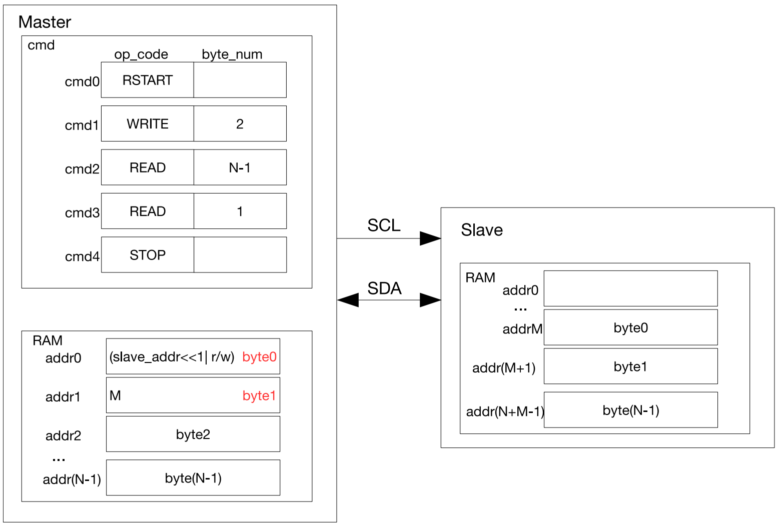


Figure 58: Master Reads N Bytes of Data from addrM in Slave with 7-bit Address

Figure 59 shows the Master reading N+M bytes of data in three segments from the Slave. The first segment shows the configuration of the cmd and the preparation of data in the Slave RAM. When the I2C_TRANS_START bit is enabled, the Master starts the operation. The Master will refresh the cmd after executing the END command. It will clear the I2C_END_DETECT_INT interrupt, set the I2C_TRANS_START bit and resume the transaction. To stop the transaction, the Master will configure the cmd, as the third segment shows, after detecting the I2C_END_DETECT_INT interrupt. After setting the I2C_TRANS_START bit, Master will send a STOP bit to stop the transaction.

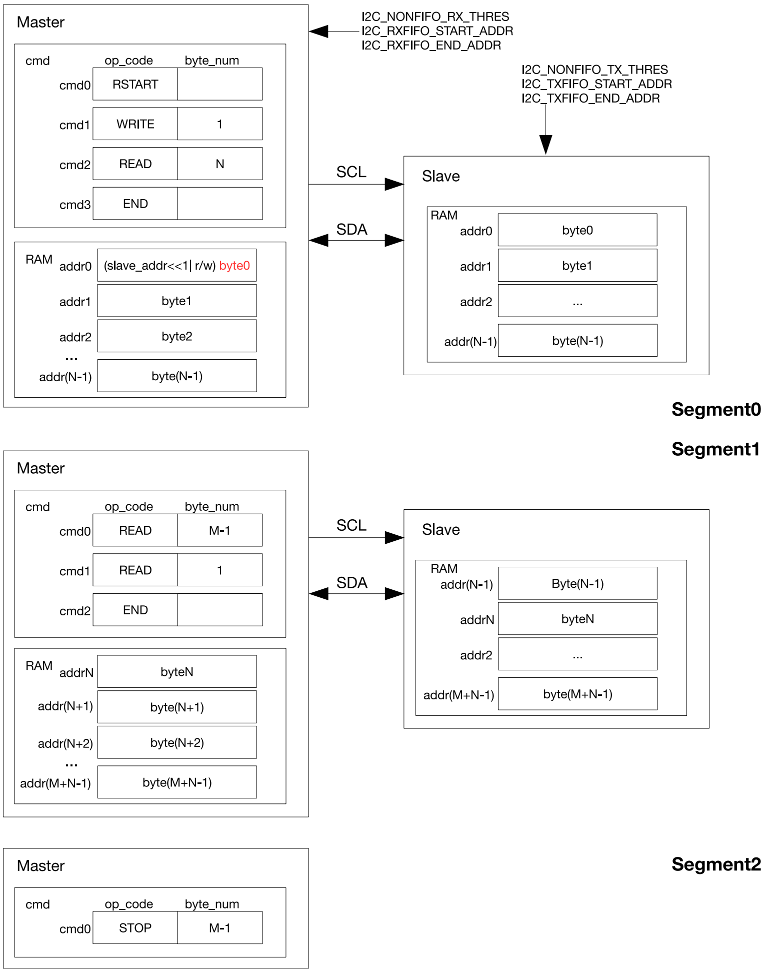


Figure 59: Master Reads from Slave with 7-bit Address in Three Segments

11.3.7 Interrupts

- I2C_TX_SEND_EMPTY_INT: Triggered when the has sent nonfifo_tx_thres bytes of data.
- I2C_RX_REC_FULL_INT: Triggered when the has received nonfifo_rx_thres bytes of data.
- I2C_ACK_ERR_INT: Triggered when the Master receives an ACK that is not as expected, or when the Slave receives an ACK whose value is 1.
- I2C_TRANS_START_INT: Triggered when the sends the START bit.
- I2C_TIME_OUT_INT: Triggered when the SCL stays high or low for more than I2C_TIME_OUT clocks.
- I2C_TRANS_COMPLETE_INT: Triggered when the detects a STOP bit.
- I2C_MASTER_TRAN_COMP_INT: Triggered when the Master sends or receives a byte.
- I2C_ARBITRATION_LOST_INT: Triggered when the Master's SCL is high, while the output value and input value of the SDA do not match.
- I2C_SLAVE_TRAN_COMP_INT: Triggered when the Slave detects a STOP bit.
- I2C_END_DETECT_INT: Triggered when the deals with the END command.

11.4 Register Summary

Name	Description	I2C0	I2C1	Acc
Configuration registers				
I2C_SLAVE_ADDR_REG	Configures the I^{2}C slave address	0x3FF53010	0x3FF67010	R/W
I2C_RXFIFO_ST_REG	FIFO status register	0x3FF53014	0x3FF67014	RO
I2C_FIFO_CONF_REG	FIFO configuration register	0x3FF53018	0x3FF67018	R/W
Timing registers				
I2C_SDA_HOLD_REG	Configures the hold time after a negative SCL edge	0x3FF53030	0x3FF67030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a positive SCL edge	0x3FF53034	0x3FF67034	R/W
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of the SCL clock	0x3FF53000	0x3FF67000	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of the SCL clock	0x3FF53038	0x3FF67038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x3FF53040	0x3FF67040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the positive edge of SCL and the negative edge of SDA	0x3FF53044	0x3FF67044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a stop condition	0x3FF53048	0x3FF67048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x3FF5304C	0x3FF6704C	R/W
Filter registers				
I2C_SCL_FILTER_CFG_REG	SCL filter configuration register	0x3FF53050	0x3FF67050	R/W
I2C_SDA_FILTER_CFG_REG	SDA filter configuration register	0x3FF53054	0x3FF67054	R/W
Interrupt registers				
I2C_INT_RAW_REG	Raw interrupt status	0x3FF53020	0x3FF67020	RO
I2C_INT_ENA_REG	Interrupt enable bits	0x3FF53028	0x3FF67028	R/W
I2C_INT_CLR_REG	Interrupt clear bits	0x3FF53024	0x3FF67024	WO
Command registers				
I2C_COMD0_REG	I^{2}C command register 0	0x3FF53058	0x3FF67058	R/W
I2C_COMD1_REG	I^{2}C command register 1	0x3FF5305C	0x3FF6705C	R/W
I2C_COMD2_REG	I^{2}C command register 2	0x3FF53060	0x3FF67060	R/W
I2C_COMD3_REG	I^{2}C command register 3	0x3FF53064	0x3FF67064	R/W
I2C_COMD4_REG	I^{2}C command register 4	0x3FF53068	0x3FF67068	R/W
I2C_COMD5_REG	I^{2}C command register 5	0x3FF5306C	0x3FF6706C	R/W
I2C_COMD6_REG	I^{2}C command register 6	0x3FF53070	0x3FF67070	R/W
I2C_COMD7_REG	I^{2}C command register 7	0x3FF53074	0x3FF67074	R/W
I2C_COMD8_REG	I^{2}C command register 8	0x3FF53078	0x3FF67078	R/W
I2C_COMD9_REG	I^{2}C command register 9	0x3FF5307C	0x3FF6707C	R/W
I2C_COMD10_REG	I^{2}C command register 10	0x3FF53080	0x3FF67080	R/W
I2C_COMD11_REG	I^{2}C command register 11	0x3FF53084	0x3FF67084	R/W
I2C_COMD12_REG	I^{2}C command register 12	0x3FF53088	0x3FF67088	R/W

Name	Description	I2C0	I2C1	Acc
I2C_COMD13_REG	I^{2}C command register 13	0x3FF5308C	0x3FF6708C	R/W
I2C_COMD14_REG	I^{2}C command register 14	0x3FF53090	0x3FF67090	R/W
I2C_COMD15_REG	I^{2}C command register 15	0x3FF53094	0x3FF67094	R/W

11.5 Registers

Register 11.1: I2C_SCL_LOW_PERIOD_REG (0x0000)

(reserved)	I2C_SCL_LOW_PERIOD	
31 … 14	13 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_LOW_PERIOD This register is used to configure for how long SCL remains low in master mode, in APB clock cycles. (R/W)

Register 11.2: I2C_CTR_REG (0x0004)

(reserved)	I2C_RX_LSB_FIRST	I2C_TX_LSB_FIRST	I2C_TRANS_START	I2C_MS_MODE	(reserved)	I2C_SAMPLE_SCL_LEVEL	I2C_SCL_FORCE_OUT	I2C_SDA_FORCE_OUT	
31 … 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	1	1	Reset

I2C_RX_LSB_FIRST This bit is used to control the storage mode for received data. (R/W)
1: receive data from the least significant bit;
0: receive data from the most significant bit.

I2C_TX_LSB_FIRST This bit is used to control the sending mode for data needing to be sent. (R/W)
1: send data from the least significant bit;
0: send data from the most significant bit.

I2C_TRANS_START Set this bit to start sending the data in txfifo. (R/W)

I2C_MS_MODE Set this bit to configure the module as an I²C Master. Clear this bit to configure the module as an I²C Slave. (R/W)

I2C_SAMPLE_SCL_LEVEL 1: sample SDA data on the SCL low level; 0: sample SDA data on the SCL high level. (R/W)

I2C_SCL_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

I2C_SDA_FORCE_OUT 0: direct output; 1: open drain output. (R/W)

Register 11.3: I2C_SR_REG (0x0008)

(reserved)	I2C_SCL_STATE_LAST	(reserved)	I2C_SCL_MAIN_STATE_LAST	I2C_TXFIFO_CNT	(reserved)	I2C_RXFIFO_CNT	(reserved)	I2C_BYTE_TRANS	I2C_SLAVE_ADDRESSED	I2C_BUS_BUSY	I2C_ARB_LOST	I2C_TIME_OUT	I2C_SLAVE_RW	I2C_ACK_REC	
31	30 28	27	26 24	23 18	17 14	13 8	7	6	5	4	3	2	1	0	
0	0 0 0	0	0 0 0	0 0 0 0 0 0	0 0 0 0	0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

I2C_SCL_STATE_LAST This field indicates the states of the state machine used to produce SCL. (RO)

0: Idle; 1: Start; 2: Negative edge; 3: Low; 4: Positive edge; 5: High; 6: Stop

I2C_SCL_MAIN_STATE_LAST This field indicates the states of the I²C module state machine. (RO)

0: Idle; 1: Address shift; 2: ACK address; 3: Rx data; 4: Tx data; 5: Send ACK; 6: Wait ACK

I2C_TXFIFO_CNT This field stores the amount of received data in RAM. (RO)

I2C_RXFIFO_CNT This field represents the amount of data needed to be sent. (RO)

I2C_BYTE_TRANS This field changes to 1 when one byte is transferred. (RO)

I2C_SLAVE_ADDRESSED When configured as an I²C Slave, and the address sent by the master is equal to the address of the slave, then this bit will be of high level. (RO)

I2C_BUS_BUSY 1: the I²C bus is busy transferring data; 0: the I²C bus is in idle state. (RO)

I2C_ARB_LOST When the I²C controller loses control of SCL line, this register changes to 1. (RO)

I2C_TIME_OUT When the I²C controller takes more than I2C_TIME_OUT clocks to receive a data bit, this field changes to 1. (RO)

I2C_SLAVE_RW When in slave mode, 1: master reads from slave; 0: master writes to slave. (RO)

I2C_ACK_REC This register stores the value of the received ACK bit. (RO)

Register 11.4: I2C_TO_REG (0x000c)

(reserved)	I2C_TIME_OUT_REG	
31 20	19 0	
0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_TIME_OUT_REG This register is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

Register 11.5: I2C_SLAVE_ADDR_REG (0x0010)

I2C_SLAVE_ADDR_10BIT_EN	(reserved)	I2C_SLAVE_ADDR	
31	30 ... 15	14 ... 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SLAVE_ADDR_10BIT_EN This field is used to enable the slave 10-bit addressing mode in master mode. (R/W)

I2C_SLAVE_ADDR When configured as an I^{2}C Slave, this field is used to configure the slave address. (R/W)

Register 11.6: I2C_RXFIFO_ST_REG (0x0014)

(reserved)	I2C_RXFIFO_TXFIFO_END_ADDR	I2C_RXFIFO_TXFIFO_START_ADDR	I2C_RXFIFO_END_ADDR	I2C_RXFIFO_START_ADDR	
31 ... 20	19 ... 15	14 ... 10	9 ... 5	4 ... 0	
0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

I2C_TXFIFO_END_ADDR This is the offset address of the last sent data, as described in nonfifo_tx_thres register. The value refreshes when I2C_TX_SEND_EMPTY_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_TXFIFO_START_ADDR This is the offset address of the first sent data, as described in nonfifo_tx_thres register. (RO)

I2C_RXFIFO_END_ADDR This is the offset address of the last received data, as described in nonfifo_rx_thres_register. This value refreshes when I2C_RX_REC_FULL_INT or I2C_TRANS_COMPLETE_INT interrupt is generated. (RO)

I2C_RXFIFO_START_ADDR This is the offset address of the last received data, as described in nonfifo_rx_thres_register. (RO)

Register 11.7: I2C_FIFO_CONF_REG (0x0018)

(reserved)	I2C_NONFIFO_TX_THRES	I2C_NONFIFO_RX_THRES	(reserved)	I2C_FIFO_ADDR_CFG_EN	I2C_NONFIFO_EN	
31 26	25 20	19 14	13 12	11	10	
0 0 0 0 0 0	0x15	0x15	0 0	0	0	Reset

I2C_NONFIFO_TX_THRES When I^{2}C sends more than nonfifo_tx_thres bytes of data, it will generate a tx_send_empty_int_raw interrupt and update the current offset address of the sent data. (R/W)

I2C_NONFIFO_RX_THRES When I^{2}C receives more than nonfifo_rx_thres bytes of data, it will generate a rx_send_full_int_raw interrupt and update the current offset address of the received data. (R/W)

I2C_FIFO_ADDR_CFG_EN When this bit is set to 1, the byte received after the I^{2}C address byte represents the offset address in the I^{2}C Slave RAM. (R/W)

I2C_NONFIFO_EN Set this bit to enable APB nonfifo access. (R/W)

Register 11.8: I2C_INT_RAW_REG (0x0020)

(reserved)	I2C_TX_SEND_EMPTY_INT_RAW	I2C_RX_REC_FULL_INT_RAW	I2C_ACK_ERR_INT_RAW	I2C_TRANS_START_INT_RAW	I2C_TIME_OUT_INT_RAW	I2C_TRANS_COMPLETE_INT_RAW	I2C_MASTER_TRAN_COMP_INT_RAW	I2C_ARBITRATION_LOST_INT_RAW	I2C_END_DETECT_INT_RAW	
31 13	12	11	10	9	8	7	6	5	3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_RAW The raw interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

I2C_RX_REC_FULL_INT_RAW The raw interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

I2C_ACK_ERR_INT_RAW The raw interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

I2C_TRANS_START_INT_RAW The raw interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_TIME_OUT_INT_RAW The raw interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_MASTER_TRAN_COMP_INT_RAW The raw interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_END_DETECT_INT_RAW The raw interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

Register 11.9: I2C_INT_CLR_REG (0x0024)

	(reserved)	I2C_TX_SEND_EMPTY_INT_CLR	I2C_RX_REC_FULL_INT_CLR	I2C_ACK_ERR_INT_CLR	I2C_TRANS_START_INT_CLR	I2C_TIME_OUT_INT_CLR	I2C_TRANS_COMPLETE_INT_CLR	I2C_MASTER_TRAN_COMP_INT_CLR	I2C_ARBITRATION_LOST_INT_CLR	I2C_END_DETECT_INT_CLR
	31 … 13	12	11	10	9	8	7	6	5	3
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0

I2C_TX_SEND_EMPTY_INT_CLR Set this bit to clear the I2C_TX_SEND_EMPTY_INT interrupt. (WO)

I2C_RX_REC_FULL_INT_CLR Set this bit to clear the I2C_RX_REC_FULL_INT interrupt. (WO)

I2C_ACK_ERR_INT_CLR Set this bit to clear the I2C_ACK_ERR_INT interrupt. (WO)

I2C_TRANS_START_INT_CLR Set this bit to clear the I2C_TRANS_START_INT interrupt. (WO)

I2C_TIME_OUT_INT_CLR Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WO)

I2C_TRANS_COMPLETE_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WO)

I2C_MASTER_TRAN_COMP_INT_CLR Set this bit to clear the I2C_MASTER_TRAN_COMP_INT interrupt. (WO)

I2C_ARBITRATION_LOST_INT_CLR Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WO)

I2C_END_DETECT_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WO)

Register 11.10: I2C_INT_ENA_REG (0x0028)

(reserved)	I2C_TX_SEND_EMPTY_INT_ENA	I2C_RX_REC_FULL_INT_ENA	I2C_ACK_ERR_INT_ENA	I2C_TRANS_START_INT_ENA	I2C_TIME_OUT_INT_ENA	I2C_TRANS_COMPLETE_INT_ENA	I2C_MASTER_TRAN_COMP_INT_ENA	I2C_ARBITRATION_LOST_INT_ENA	I2C_END_DETECT_INT_ENA	
31 13	12	11	10	9	8	7	6	5	3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_ENA The interrupt enable bit for the I2C_TX_SEND_EMPTY_INT interrupt. (R/W)

I2C_RX_REC_FULL_INT_ENA The interrupt enable bit for the I2C_RX_REC_FULL_INT interrupt. (R/W)

I2C_ACK_ERR_INT_ENA The interrupt enable bit for the I2C_ACK_ERR_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA The interrupt enable bit for the I2C_TRANS_START_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA The interrupt enable bit for the I2C_TIME_OUT_INT interrupt. (R/W)

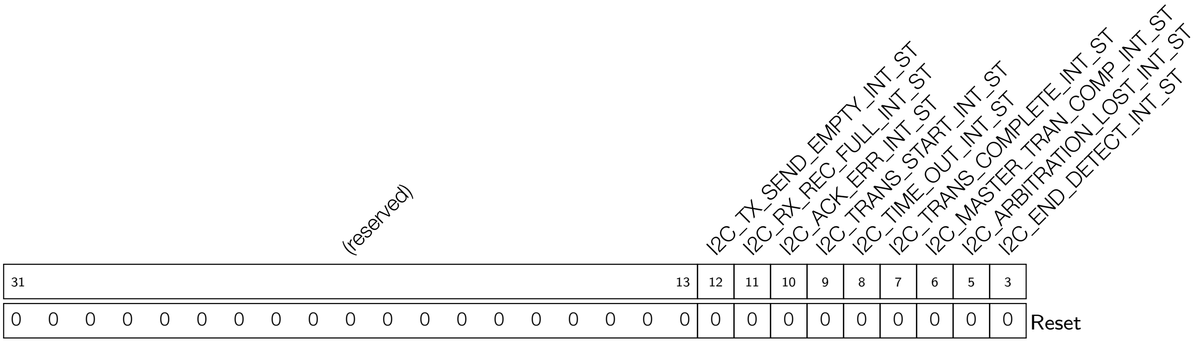
I2C_TRANS_COMPLETE_INT_ENA The interrupt enable bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_MASTER_TRAN_COMP_INT_ENA The interrupt enable bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA The interrupt enable bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA The interrupt enable bit for the I2C_END_DETECT_INT interrupt. (R/W)

Register 11.11: I2C_INT_STATUS_REG (0x002c)



31 … 13	12	11	10	9	8	7	6	5	3	
(reserved)	I2C_TX_SEND_EMPTY_INT_ST	I2C_RX_REC_FULL_INT_ST	I2C_ACK_ERR_INT_ST	I2C_TRANS_START_INT_ST	I2C_TIME_OUT_INT_ST	I2C_TRANS_COMPLETE_INT_ST	I2C_MASTER_TRAN_COMP_INT_ST	I2C_ARBITRATION_LOST_INT_ST	I2C_END_DETECT_INT_ST	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_ST The masked interrupt status bit for the I2C_TX_SEND_EMPTY_INT interrupt. (RO)

I2C_RX_REC_FULL_INT_ST The masked interrupt status bit for the I2C_RX_REC_FULL_INT interrupt. (RO)

I2C_ACK_ERR_INT_ST The masked interrupt status bit for the I2C_ACK_ERR_INT interrupt. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

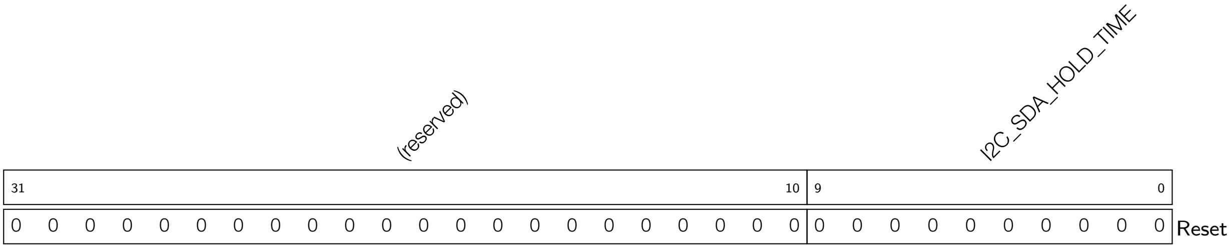
I2C_TRANS_COMPLETE_INT_ST The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_MASTER_TRAN_COMP_INT_ST The masked interrupt status bit for the I2C_MASTER_TRAN_COMP_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

Register 11.12: I2C_SDA_HOLD_REG (0x0030)



31 … 10	9 … 0	
(reserved)	I2C_SDA_HOLD_TIME	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SDA_HOLD_TIME This register is used to configure the time to hold the data after the negative edge of SCL, in APB clock cycles. (R/W)

Register 11.13: I2C_SDA_SAMPLE_REG (0x0034)

(reserved)	I2C_SDA_SAMPLE_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SDA_SAMPLE_TIME This register is used to configure for how long SDA is sampled, in APB clock cycles. (R/W)

Register 11.14: I2C_SCL_HIGH_PERIOD_REG (0x0038)

(reserved)	I2C_SCL_HIGH_PERIOD	
31 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_HIGH_PERIOD This register is used to configure for how long SCL remains high in master mode, in APB clock cycles. (R/W)

Register 11.15: I2C_SCL_START_HOLD_REG (0x0040)

(reserved)	I2C_SCL_START_HOLD_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0	Reset

I2C_SCL_START_HOLD_TIME This register is used to configure the time between the negative edge of SDA and the negative edge of SCL for a START condition, in APB clock cycles. (R/W)

Register 11.16: I2C_SCL_RSTART_SETUP_REG (0x0044)

(reserved)	I2C_SCL_RSTART_SETUP_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0	Reset

I2C_SCL_RSTART_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the negative edge of SDA for a RESTART condition, in APB clock cycles. (R/W)

Register 11.17: I2C_SCL_STOP_HOLD_REG (0x0048)

(reserved)	I2C_SCL_STOP_HOLD_TIME	
31 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_HOLD_TIME This register is used to configure the delay after the STOP condition, in APB clock cycles. (R/W)

Register 11.18: I2C_SCL_STOP_SETUP_REG (0x004C)

(reserved)	I2C_SCL_STOP_SETUP_TIME	
31 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_SETUP_TIME This register is used to configure the time between the positive edge of SCL and the positive edge of SDA, in APB clock cycles. (R/W)

Register 11.19: I2C_SCL_FILTER_CFG_REG (0x0050)

	(reserved)	I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	
	31 4	3	2 0	
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SCL_FILTER_EN This is the filter enable bit for SCL. (R/W)

I2C_SCL_FILTER_THRES When a pulse on the SCL input has smaller width than this register value in APB clock cycles, the I^{2}C controller will ignore that pulse. (R/W)

Register 11.20: I2C_SDA_FILTER_CFG_REG (0x0054)

	(reserved)	I2C_SDA_FILTER_EN	I2C_SDA_FILTER_THRES	
	31 4	3	2 0	
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SDA_FILTER_EN This is the filter enable bit for SDA. (R/W)

I2C_SDA_FILTER_THRES When a pulse on the SDA input has smaller width than this register value in APB clock cycles, the I^{2}C controller will ignore that pulse. (R/W)

Register 11.21: I2C_COMD*n*_REG (*n*: 0-15) (0x58+4**n*)

	I2C_COMMAND*n*_DONE	(reserved)	I2C_COMMAND*n*	
	31	30 14	13 0	
	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_COMMAND*n*_DONE When command *n* is done in I^{2}C Master mode, this bit changes to high level. (R/W)

I2C_COMMAND*n* This is the content of command *n*. It consists of three parts: (R/W)

op_code is the command, 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END.

Byte_num represents the number of bytes that need to be sent or received.

ack_check_en, ack_exp and ack are used to control the ACK bit. See I^{2}C cmd structure for more information.

12. I2S

12.1 Overview

The I2S bus provides a flexible communication interface for streaming digital data in multimedia applications, especially digital audio applications. The ESP32 includes two I2S interfaces: I2S0 and I2S1.

The I2S standard bus defines three signals: a clock signal, a channel selection signal, and a serial data signal. A basic I2S data bus has one master and one slave. The roles remain unchanged throughout the communication. The I2S modules on the ESP32 provide separate transmit and receive channels for high performance.

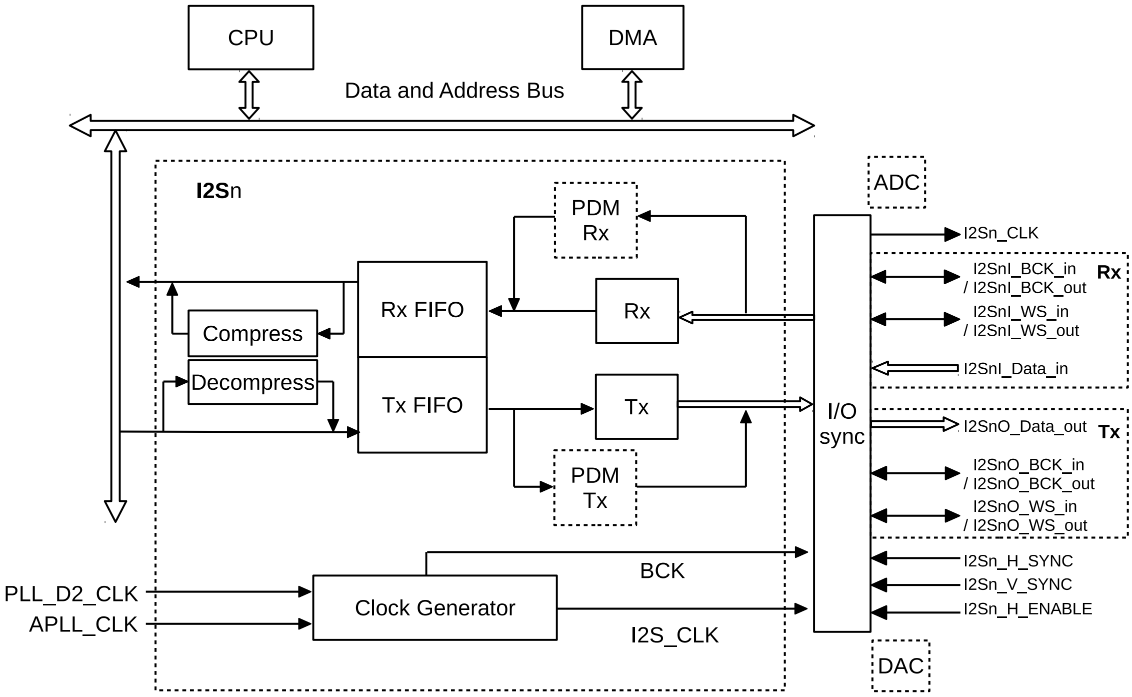


Figure 60: I2S System Block Diagram

Figure 60 is the system block diagram of the ESP32 I2S module. In the figure above, the value of ”*n*” can be either 0 or 1. There are two independent I2S modules embedded in ESP32, namely I2S0 and I2S1. Each I2S module contains a Tx (transmit) unit and a Rx (receive) unit. Both the Tx unit and the Rx unit have a three-wire interface that includes a clock line, a channel selection line and a serial data line. The serial data line of the Tx unit is fixed as output, and the serial data line of the receive unit is fixed as input. The clock line and the channel selection line of the Tx and Rx units can be configured to both master transmitting mode and slave receiving mode. In the LCD mode, the serial data line extends to the parallel data bus. Both the Tx unit and the Rx unit have a 32-bit-wide FIFO with a depth of 64. Besides, only I2S0 supports on-chip DAC/ADC modes, as well as receiving and transmitting PDM signals.

The right side of Figure 60 shows the signal bus of the I2S module. The signal naming rule of the Rx and Tx units is I2S*n*A_B_C, where ”*n*” stands for either I2S0 or I2S1; ”A” represents the direction of I2S module’s data bus signal, ”I” represents input, ”O” represents output; ”B” represents signal function; ”C” represents the signal direction, ”in” means that the signal is input into the I2S module, while ”out” means that the I2S module outputs the signal. For a detailed description of the I2S signal bus, please refer to Table 56.

Table 56: I2S Signal Bus Description

Signal Bus	Signal Direction	Data Signal Direction
I2S*n*I_BCK_in	In slave mode, I2S module accepts signals.	I2S module receives data.
I2S*n*I_BCK_out	In master mode, I2S module outputs signals.	I2S module receives data.
I2S*n*I_WS_in	In slave mode, I2S module accepts signals.	I2S module receives data.
I2S*n*I_WS_out	In master mode, I2S module outputs signals.	I2S module receives data.
I2S*n*I_Data_in	I2S module accepts signals.	In I2S mode, I2S*n*I_Data_in[15] is the serial data bus of I2S. In LCD mode, the data bus width can be configured as needed.
I2S*n*O_Data_out	I2S module outputs signals.	In I2S mode, I2S*n*O_Data_out[23] is the serial data bus of I2S. In LCD mode, the data bus width can be configured as needed.
I2S*n*O_BCK_in	In slave mode, I2S module accepts signals.	I2S module sends data.
I2S*n*O_BCK_out	In master mode, I2S module outputs signals.	I2S module sends data.
I2S*n*O_WS_in	In slave mode, I2S module accepts signals.	I2S module sends data.
I2S*n*O_WS_out	In master mode, I2S module outputs signals.	I2S module sends data.
I2S*n*_CLK	I2S module outputs signals.	It is used as a clock source for peripheral chips.
I2S*n*_H_SYNC	In Camera mode, I2S module accepts signals.	The signals are sent from the Camera.
I2S*n*_V_SYNC		
I2S*n*_H_ENABLE		

Table 56 describes the signal bus of the I2S module. Except for the I2S*n*_CLK signal, all other signals are mapped to the chip pin via the GPIO matrix and IO MUX. The I2S*n*_CLK signal is mapped to the chip pin via the IO_MUX. For details, please refer to the chapter about IO_MUX and the GPIO Matrix.

12.2 Features

I2S mode

- Configurable high-precision output clock
- Full-duplex and half-duplex data transmit and receive modes
- Supports multiple digital audio standards
- Embedded A-law compression/decompression module
- Configurable clock signal
- Supports PDM signal input and output
- Configurable data transmit and receive modes

LCD mode

- Supports multiple LCD modes, including external LCD
- Supports external Camera

- Supports on-chip DAC/ADC modes

I2S interrupts

- Standard I2S interface interrupts
- I2S DMA interface interrupts

12.3 The Clock of I2S Module

As is shown in Figure 61, I2S*n*_CLK, as the master clock of I2S module, is derived from the 160 MHz clock PLL_D2_CLK or the configurable analog PLL output clock APLL_CLK. The serial clock (BCK) of the I2S module is derived from I2S*n*_CLK. The I2S_CLKA_ENA bit of register I2S_CLKM_CONF_REG is used to select either PLL_D2_CLK or APLL_CLK as the clock source for I2S*n*. PLL_D2_CLK is used as the clock source for I2S*n*, by default.

Notice:

- When using PLL_D2_CLK as the clock source, it is not recommended to divide it using decimals. For high performance audio applications, the analog PLL output clock source APLL_CLK must be used to acquire highly accurate I2S*n*_CLK and BCK. For further details, please refer to the chapter entitled Reset and Clock.
- When ESP32 I2S works in slave mode, the master must use I2S*n*_CLK as the master clock and f_{i2s} >= 8 * f_{BCK}.

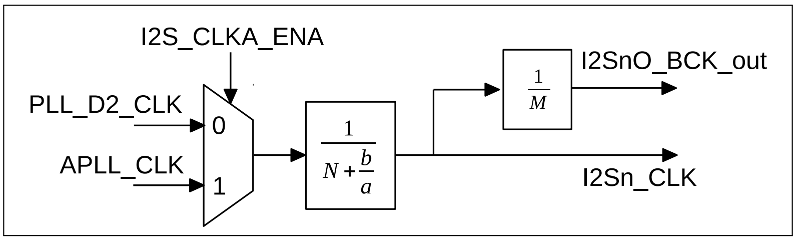


Figure 61: I2S Clock

The relation between I2S*n*_CLK frequency f_{i2s} and the divider clock source frequency f_{pll} can be seen in the equation below:

$$f_{\text{i2s}} = \frac{f_{\text{pll}}}{\mathrm{N} + \frac{\mathrm{b}}{\mathrm{a}}}$$

”N”, whose value is >=2, corresponds to the REG _CLKM_DIV_NUM [7: 0] bits of register I2S_CLKM_CONF_REG , ”b” is the I2S_CLKM_DIV_B[5:0] bit and ”a” is the I2S_CLKM_DIV_A[5:0] bit.

In master mode, the serial clock BCK in the I2S module is derived from I2S*n*_CLK, that is:

$$f_{\text{BCK}} = \frac{f_{\text{i2s}}}{\mathrm{M}}$$

In master transmitting mode, ”M”, whose value is >=2, is the I2S_TX_BCK_DIV_NUM[5:0] bit of register I2S_SAMPLE_RATE_CONF_REG. In master receiving mode, ”M” is the I2S_RX_BCK_DIV_NUM[5:0] bit of register I2S_SAMPLE_RATE_CONF_REG.

12.4 I2S Mode

The ESP32 I2S module integrates an A-law compression/decompression module to enable compression/decompression of the received audio data. The RX_PCM_BYPASS bit and the TX_PCM_BYPASS bit of register I2S_CONF1_REG should be cleared when using the A-law compression/decompression module.

12.4.1 Supported Audio Standards

In the I2S bus, BCK is the serial clock, WS is the left- /right-channel selection signal (also called word select signal), and SD is the serial data signal for transmitting/receiving digital audio data. WS and SD signals in the I2S module change on the falling edge of BCK, while the SD signal can be sampled on the rising edge of BCK. If the I2S_RX_RIGHT_FIRST bit and the I2S_TX_RIGHT_FIRST bit of register I2S_CONF_REG are set to 1, the I2S module is configured to receive and transmit right-channel data first. Otherwise, the I2S module receives and transmits left-channel data first.

12.4.1.1 Philips Standard

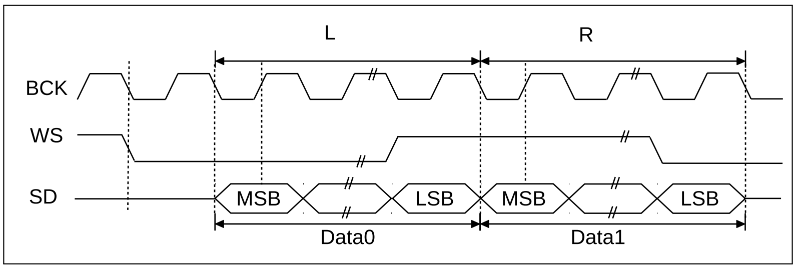


Figure 62: Philips Standard

As is shown in Figure 62, the Philips I2S bus specifications require that the WS signal starts to change a BCK clock cycle earlier than the SD signal, which means that the WS signal takes effect a clock cycle before the first bit of the current channel-data transmission, while the WS signal continues until the end of the current channel-data transmission. The SD signal line transmits the most significant bit of audio data first. If the I2S_RX_MSB_SHIFT bit and the I2S_TX_MSB_SHIFT bit of register I2S_CONF_REG are set to 1, respectively, the I2S module will use the Philips standard when receiving and transmitting data.

12.4.1.2 MSB Alignment Standard

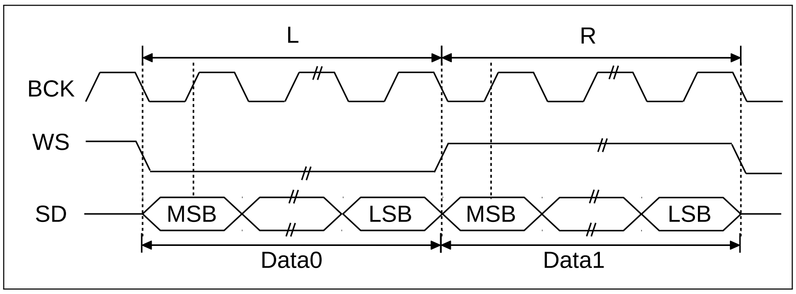


Figure 63: MSB Alignment Standard

The MSB alignment standard is shown in Figure 63. WS and SD signals both change simultaneously on the falling edge of BCK under the MSB alignment standard. The WS signal continues until the end of the current channel-data transmission, and the SD signal line transmits the most significant bit of audio data first. If the I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT bits of register I2S_CONF_REG are cleared, the I2S module will use the MSB alignment standard when receiving and transmitting data.

12.4.1.3 PCM Standard

As is shown in Figure 64, under the short frame synchronization mode of the PCM standard, the WS signal starts to change a BCK clock cycle earlier than the SD signal, which means that the WS signal takes effect a clock cycle earlier than the first bit of the current channel-data transmission and continues for one extra BCK clock cycle. The SD signal line transmits the most significant bit of audio data first. If the I2S_RX_SHORT_SYNC and I2S_TX_SHORT_SYNC bits of register I2S_CONF_REG are set, the I2S module will receive and transmit data in the short frame synchronization mode.

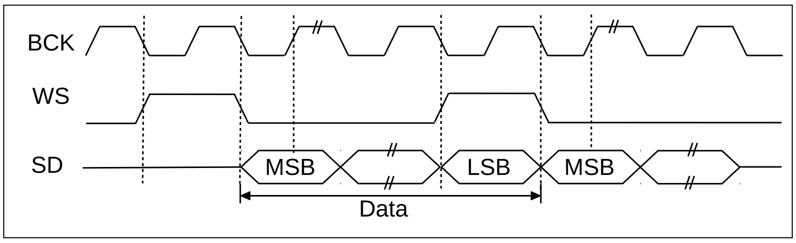


Figure 64: PCM Standard

12.4.2 Module Reset

The four low-order bits in register I2S_CONF_REG, that is, I2S_TX_RESET, I2S_RX_RESET, I2S_TX_FIFO_RESET and I2S_RX_FIFO_RESET reset the receive module, the transmit module and the corresponding FIFO buffer, respectively. In order to finish a reset operation, the corresponding bit should be set and then cleared by software.

12.4.3 FIFO Operation

The data read/write packet length for a FIFO operation is 32 bits. The data packet format for the FIFO buffer can be configured using configuration registers. As shown in Figure 60, both sent and received data should be written into FIFO first and then read from FIFO. There are two approaches to accessing the FIFO; one is to directly access the FIFO using a CPU, the other is to access the FIFO using a DMA controller.

Generally, both the I2S_RX_FIFO_MOD_FORCE_EN bit and I2S_TX_FIFO_MOD_FORCE_EN bits of register I2S_FIFO_CONF_REG should be set to 1. I2S_TX_DATA_NUM[5:0] bit and I2S_RX_DATA_NUM[5:0] are used to control the length of the data that have been sent, received and buffered. Hardware inspects the received-data length RX_LEN and the transmitted-data length TX_LEN. Both the received and the transmitted data are buffered in the FIFO method.

When RX_LEN is greater than I2S_RX_DATA_NUM[5:0], the received data, which is buffered in FIFO, has reached the set threshold and needs to be read out to prevent an overflow. When TX_LEN is less than I2S_TX_DATA_NUM[5:0], the transmitted data, which is buffered in FIFO, has not reached the set threshold and software can continue feeding data into FIFO.

12.4.4 Sending Data

The ESP32 I2S module carries out a data-transmit operation in three stages:

- Read data from internal storage and transfer it to FIFO
- Read data to be sent from FIFO
- Clock out data serially, or in parallel, as configured by the user

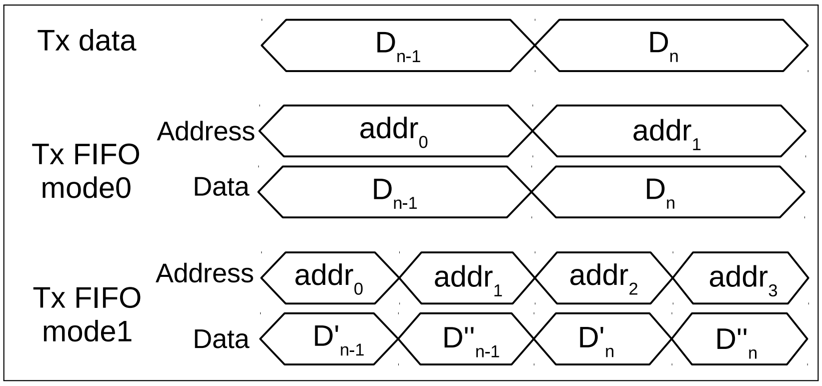


Figure 65: Tx FIFO Data Mode

Table 57: Register Configuration

	I2S_TX_FIFO_MOD[2:0]	Description
Tx FIFO mode0	0	16-bit dual channel data
	2	32-bit dual channel data
	3	32-bit single channel data
Tx FIFO mode1	1	16-bit single channel data

At the first stage, there are two modes for data to be sent and written into FIFO. In Tx FIFO mode0, the Tx data-to-be-sent are written into FIFO according to the time order. In Tx FIFO mode1, the data-to-be-sent are divided into 16 high- and 16 low-order bits. Then, both the 16 high- and 16 low-order bits are recomposed and written into FIFO. The details are shown in Figure 65 with the corresponding registers listed in Table 57. $D_n^{'}$ consists of 16 high-order bits of D_n and 16 zeros. $D_n^{''}$ consists of 16 low-order bits of D_n and 16 zeros. That is to say, $D_n^{'} = \{D_n[31:16], 16'h0\}$, $D_n^{''} = \{D_n[15:0], 16'h0\}$.

At the second stage, the system reads data that will be sent from FIFO, according to the relevant register configuration. The mode in which the system reads data from FIFO is relevant to the configuration of I2S_TX_FIFO_MOD[2.0] and I2S_TX_CHAN_MOD[2:0]. I2S_TX_FIFO_MOD[2.0] determines whether the data are 16-bit or 32-bit, as shown in Table 57, while I2S_TX_CHAN_MOD[2:0] determines the format of the data-to-be-sent, as shown in Table 58.

Table 58: Send Channel Mode

I2S_TX_CHAN_MOD[2:0]	Description
0	Dual channel mode
1	Mono mode When I2S_TX_MSB_RIGHT equals 0, the left-channel data are ”holding” their values and the right-channel data change into the left-channel data.

I2S_TX_CHAN_MOD[2:0]	Description
	When I2S_TX_MSB_RIGHT equals 1, the right-channel data are ”holding” their values and the left-channel data change into the right-channel data.
2	Mono mode When I2S_TX_MSB_RIGHT equals 0, the right-channel data are ”holding” their values and the left-channel data change into the right-channel data. When I2S_TX_MSB_RIGHT equals 1, the left-channel data are ”holding” their values and the right-channel data change into the left-channel data.
3	Mono mode When I2S_TX_MSB_RIGHT equals 0, the left-channel data are constants in the range of REG[31:0]. When I2S_TX_MSB_RIGHT equals 1, the right-channel data are constants in the range of REG[31:0].
4	Mono mode When I2S_TX_MSB_RIGHT equals 0, the right-channel data are constants in the range of REG[31:0]. When I2S_TX_MSB_RIGHT equals 1, the left-channel data are constants in the range of REG[31:0].

REG[31:0] is the value of register I2S_CONF_SINGLE_DATA_REG[31:0].

The output of the third stage is determined by the mode of the I2S and I2S_TX_BITS_MOD[5:0] bits of register I2S_SAMPLE_RATE_CONF_REG.

12.4.5 Receiving Data

The data-receive phase of the ESP32 I2S module consists of another three stages:

- The input serial-bit stream is transformed into a 64-bit parallel-data stream in I2S mode. In LCD mode, the input parallel-data stream will be extended to a 64-bit parallel-data stream.
- Received data are written into FIFO.
- Data are read from FIFO by CPU/DMA and written into the internal memory.

At the first stage of receiving data, the received-data stream is expanded to a zero-padded parallel-data stream with 32 high-order bits and 32 low-order bits, according to the level of the I2S*n*I_WS_out (or I2S*n*I_WS_in) signal. The I2S_RX_MSB_RIGHT bit of register I2S_CONF_REG is used to determine how the data are to be expanded.

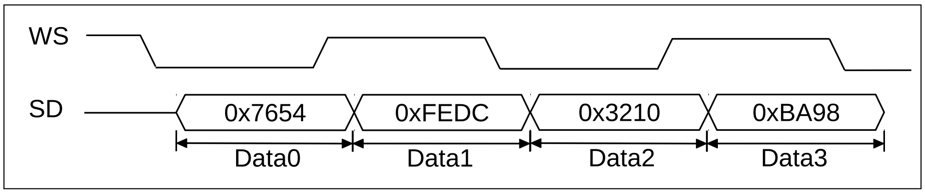


Figure 66: The First Stage of Receiving Data

For example, as is shown in Figure 66, if the width of serial data is 16 bits, when I2S_RX_RIGHT_FIRST equals 1, Data0 will be discarded and I2S will start receiving data from Data1. If I2S_RX_MSB_RIGHT equals 1, data of the first stage would be $\{0xFEDC0000, 0x32100000\}$. If I2S_RX_MSB_RIGHT equals 0, data of the first stage would be $\{0x32100000, 0xFEDC0000\}$. When I2S_RX_RIGHT_FIRST equals 0, I2S will start receiving data from Data0. If I2S_RX_MSB_RIGHT equals 1, data of the first stage would be $\{0xFEDC0000, 0x76540000\}$. If I2S_RX_MSB_RIGHT equals 0, data of the first stage would be $\{0x76540000, 0xFEDC0000\}$.

As is shown in Table 59 and Figure 67, at the second stage, the received data of the Rx unit is written into FIFO. There are four modes of writing received data into FIFO. Each mode corresponds to a value of I2S_RX_FIFO_MOD[2:0] bit.

Table 59: Modes of Writing Received Data into FIFO and the Corresponding Register Configuration

I2S_RX_FIFO_MOD[2:0]	Data format
0	16-bit dual channel data
1	16-bit single channel data
2	32-bit dual channel data
3	32-bit single channel data

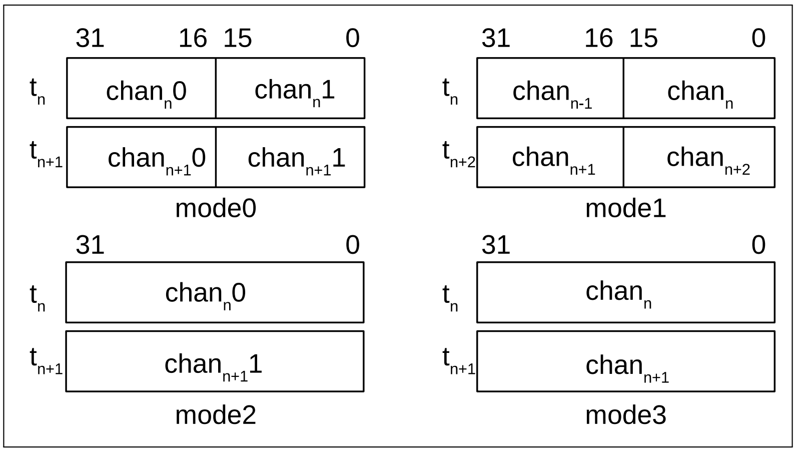


Figure 67: Modes of Writing Received Data into FIFO

At the third stage, CPU or DMA will read data from FIFO and write them into the internal memory directly. The register configuration that each mode corresponds to is shown in Table 60.

Table 60: The Register Configuration to Which the Four Modes Correspond

I2S_RX_MSB_RIGHT	I2S_RX_CHAN_MOD	mode0	mode1	mode2	mode3
0	0	left channel + right channel	-	left channel + right channel	-
	1		left channel + left channel		left channel + left channel
	2		right channel + right channel		right channel + right channel
	3		-		-
1	0	right channel + left channel	-	right channel + left channel	-
	1		right channel + right channel		right channel + right channel
	2		left channel + left channel		left channel + left channel
	3		-		-

12.4.6 I2S Master/Slave Mode

The ESP32 I2S module can be configured to act as a master or slave device on the I2S bus. The module supports slave transmitter and receiver configurations in addition to master transmitter and receiver configurations. All these modes can support full-duplex and half-duplex communication over the I2S bus.

I2S_RX_SLAVE_MOD bit and I2S_TX_SLAVE_MOD bit of register I2S_CONF_REG can configure I2S to slave receiving mode and slave transmitting mode, respectively.

I2S_TX_START bit of register I2S_CONF_REG is used to enable transmission. When I2S is in master transmitting mode and this bit is set, the module will keep driving the clock signal and data of left and right channels. If FIFO sends out all the buffered data and there are no new data to shift, the last batch of data will be looped on the data line. When this bit is reset, master will stop driving clock and data lines. When I2S is configured to slave transmitting mode and this bit is set, the module will wait for the master BCK clock to enable a transmit operation.

The I2S_RX_START bit of register I2S_CONF_REG is used to enable a receive operation. When I2S is in master transmitting mode and this bit is set, the module will keep driving the clock signal and sampling the input data stream until this bit is reset. If I2S is configured to slave receiving mode and this bit is set, the receiving module will wait for the master BCK clock to enable a receiving operation.

12.4.7 I2S PDM

As is shown in Figure 60, ESP32 I2S0 allows for pulse density modulation (PDM), which enables fast conversion between pulse code modulation (PCM) and PDM signals.

The output clock of PDM is mapped to the I2S0*_WS_out signal. Its configuration is identical to I2S's BCK. Please refer to section 12.3, "The Clock of I2S Module", for further details. The bit width for both received and transmitted I2S PCM signals is 16 bits.

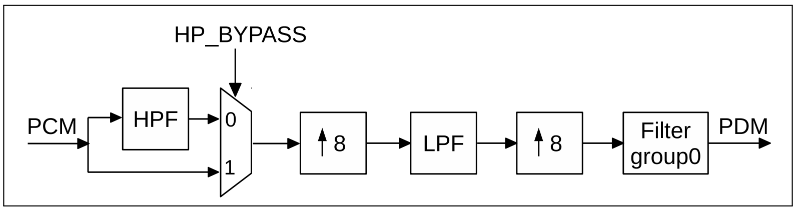


Figure 68: PDM Transmitting Module

The PDM transmitting module is used to convert PCM signals into PDM signals, as shown in Figure 68. HPF is a high-speed channel filter, and LPF is a low-speed channel filter. The PDM signal is derived from the PCM signal, after upsampling and filtering. Signal I2S_TX_PDM_HP_BYPASS of register I2S_PDM_CONF_REG can be set to bypass the HPF at the PCM input. Filter module group0 carries out the upsampling. If the frequency of the PDM signal is f_{pdm} and the frequency of the PCM signal is f_{pcm}, the relation between f_{pdm} and f_{pcm} is given by:

$$f_{\text{pdm}} = 64 \times f_{\text{pcm}} \times \frac{I2S\_TX\_PDM\_FP}{I2S\_TX\_PDM\_FS}$$

The upsampling factor of 64 is the result of the two upsampling stages.

Table 61 lists the configuration rates of the I2S_TX_PDM_FP bit and the I2S_TX_PDM_FS bit of register I2S_PDM_FREQ_CONF_REG, whose output PDM signal frequency remains 48×128 KHz at different PCM signal frequencies.

Table 61: Upsampling Rate Configuration

f_{pcm} (KHz)	I2S_TX_PDM_FP	I2S_TX_PDM_FS	f_{pdm} (KHz)
48	960	480	48×128
44.1	960	441	
32	960	320	
24	960	240	
16	960	160	
8	960	80	

The I2S_TX_PDM_SINC_OSR2 bit of I2S_PDM_CONF_REG is the upsampling rate of the Filter group0.

$$I2S\_TX\_PDM\_SINC\_OSR2 = \left\lfloor \frac{I2S\_TX\_PDM\_FP}{I2S\_TX\_PDM\_FS} \right\rfloor$$

As is shown in Figure 69, the I2S_TX_PDM_EN bit and the I2S_PCM2PDM_CONV_EN bit of register I2S_PDM_CONF_REG should be set to 1 to use the PDM sending module. The I2S_TX_PDM_SIGMADELTA_IN_SHIFT bit, I2S_TX_PDM_SINC_IN_SHIFT bit, I2S_TX_PDM_LP_IN_SHIFT bit and I2S_TX_PDM_HP_IN_SHIFT bit of register I2S_PDM_CONF_REG are used to adjust the size of the input signal of each filter module.

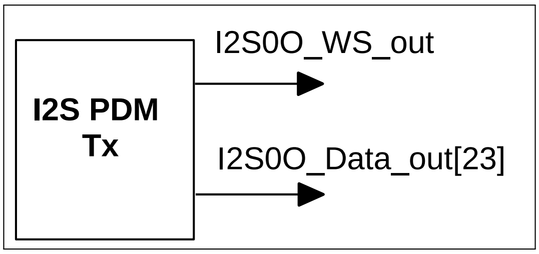


Figure 69: PDM Sends Signal

As is shown in Figure 70, the I2S_RX_PDM_EN bit and the I2S_PDM2PCM_CONV_EN bit of register I2S_PDM_CONF_REG should be set to 1, in order to use the PDM receiving module. As is shown in Figure 71, the PDM receiving module will convert the received PDM signal into a 16-bit PCM signal. Filter group1 is used to downsample the PDM signal, and the I2S_RX_PDM_SINC_DSR_16_EN bit of register I2S_PDM_CONF_REG is used to adjust the corresponding down-sampling rate.

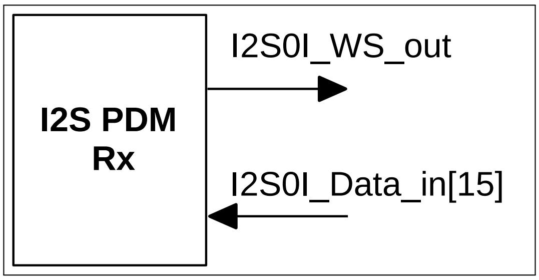


Figure 70: PDM Receives Signal

Table 62 shows the configuration of the I2S_RX_PDM_SINC_DSR_16_EN bit whose PCM signal frequency remains 48 KHz at different PDM signal frequencies.

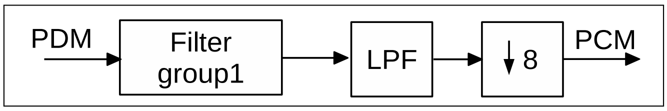


Figure 71: PDM Receive Module

Table 62: Down-sampling Configuration

PDM freq (KHz)	I2S_RX_PDM_SINC_DSR_16_EN	PCM freq (KHz)
$f_{\text{pcm}}\times128$	1	f_{pcm}
$f_{\text{pcm}}\times64$	0	

12.5 LCD Mode

There are three operational modes in the LCD mode of ESP32 I2S:

- LCD master transmitting mode
- Camera slave receiving mode
- ADC/DAC mode

The clock configuration of the LCD master transmitting mode is identical to I2S's clock configuration. In the LCD mode, the frequency of WS is half of f_{BCK}.

In the ADC/DAC mode, use PLL_D2_CLK as the clock source.

12.5.1 LCD Master Transmitting Mode

As is shown in Figure 72, the WR signal of LCD connects to the WS signal of I2S. The LCD data bus width is 24 bits.

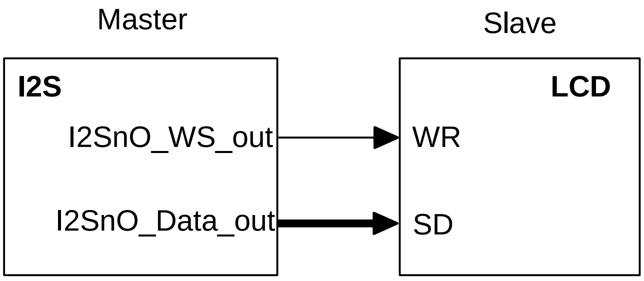


Figure 72: LCD Master Transmitting Mode

The I2S_LCD_EN bit of register I2S_CONF2_REG needs to be set and the I2S_TX_SLAVE_MOD bit of register I2S_CONF_REG needs to be cleared, in order to configure I2S to the LCD master transmitting mode. Meanwhile, data should be sent under the correct mode, according to the I2S_TX_CHAN_MOD[2:0] bit of register I2S_CONF_CHAN_REG and the I2S_TX_FIFO_MOD[2:0] bit of register I2S_FIFO_CONF_REG. The WS signal needs to be inverted when it is routed through the GPIO Matrix. For details, please refer to the chapter about IO_MUX and the GPIO Matrix. The I2S_LCD_TX_SDX2_EN bit and the I2S_LCD_TX_WRX2_EN bit of register I2S_CONF2_REG should be set to the LCD master transmitting mode, so that both the data bus and WR signal work in the appropriate mode.

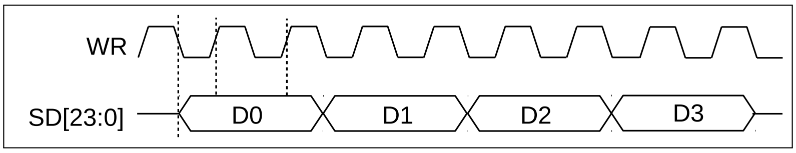


Figure 73: LCD Master Transmitting Data Frame, Form 1

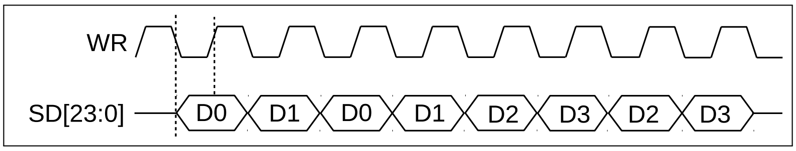


Figure 74: LCD Master Transmitting Data Frame, Form 2

As is shown in Figure 73 and Figure 74, the I2S_LCD_TX_WRX2_EN bit should be set to 1 and the I2S_LCD_TX_SDX2_EN bit should be set to 0 in the data frame, form 1. Both I2S_LCD_TX_SDX2_EN bit and I2S_LCD_TX_WRX2_EN bit are set to 1 in the data frame, form 2.

12.5.2 Camera Slave Receiving Mode

ESP32 I2S supports a camera slave mode for high-speed data transfer from external camera modules. As shown in Figure 75, in this mode, I2S is set to slave receiving mode. Besides the 16-channel data signal bus I2S*n*I_Data_in, there are other signals, such as I2S*n*_H_SYNC, I2S*n*_V_SYNC and I2S*n*_H_ENABLE.

The PCLK in the Camera module connects to I2S*n*I_WS_in in the I2S module, as Figure 75 shows.

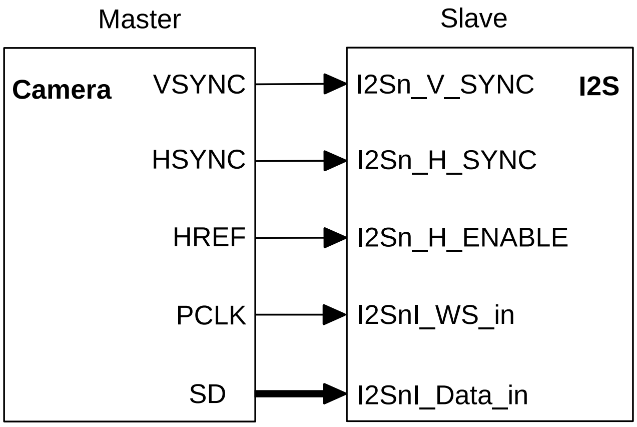


Figure 75: Camera Slave Receiving Mode

When I2S is in the camera slave receiving mode, and when I2S*n*_H_SYNC, I2S_V_SYNC and I2S_H_REF are held high, the master starts transmitting data, that is,

$$transmission\_start = (I2Sn\_H\_SYNC == 1)\&\&(I2Sn\_V\_SYNC == 1)\&\&(I2Sn\_H\_ENABLE == 1)$$

Thus, during data transmission, these three signals should be kept at a high level. For example, if the I2S*n*_V_SYNC signal of a camera is at low level during data transmission, it will be inverted when routed to the I2S module. ESP32 supports signal inversion through the GPIO matrix. For details, please refer to the chapter about IO_MUX and the GPIO Matrix.

In order to make I2S work in camera mode, the I2S_LCD_EN bit and the I2S_CAMERA_EN bit of register I2S_CONF2_REG are set to 1, the I2S_RX_SLAVE_MOD bit of register I2S_CONF_REG is set to 1, the I2S_RX_MSB_RIGHT bit and the I2S_RX_RIGHT_FIRST bit of I2S_CONF_REG are set to 0. Thus, I2S works in the LCD slave receiving mode. At the same time, in order to use the correct mode to receive data, both the I2S_RX_CHAN_MOD[2:0] bit of register I2S_CONF_CHAN_REG and the I2S_RX_FIFO_MOD[2:0] bit of register I2S_FIFO_CONF_REG are set to 1.

12.5.3 ADC/DAC mode

In LCD mode, ESP32's ADC and DAC can receive data. When the I2S0 module connects to the on-chip ADC, the I2S0 module should be set to master receiving mode. Figure 76 shows the signal connection between the I2S0 module and the ADC.

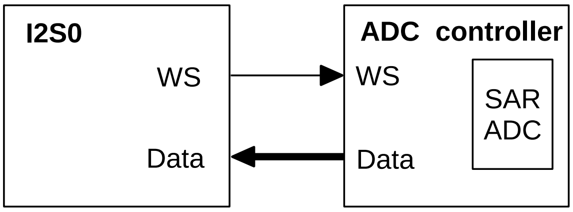


Figure 76: ADC Interface of I2S0

Firstly, the I2S_LCD_EN bit of register I2S_CONF2_REG is set to 1, and the I2S_RX_SLAVE_MOD bit of register I2S_CONF_REG is set to 0, so that the I2S0 module works in LCD master receiving mode, and the I2S0 module clock is configured such that the WS signal of I2S0 outputs an appropriate frequency. Then, the APB_CTRL_SARADC_DATA_TO_I2S bit of register APB_CTRL_APB_SARADC_CTRL_REG is set to 1. Enable I2S to receive data after configuring the relevant registers of SARADC. For details, please refer to Chapter On-Chip Sensors and Analog Signal Processing.

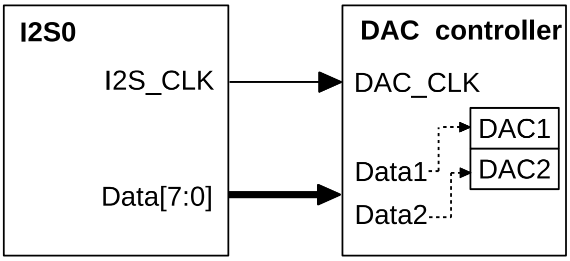


Figure 77: DAC Interface of I2S

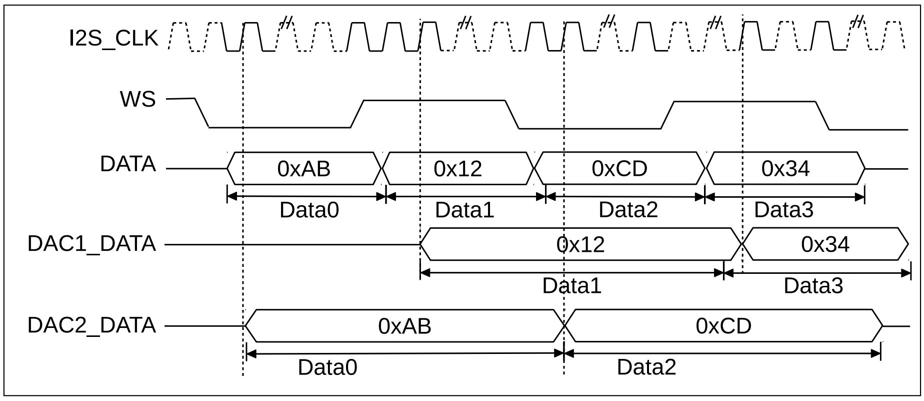


Figure 78: Data Input by I2S DAC Interface

The I2S0 module should be configured to master transmitting mode when it connects to the on-chip DAC. Figure 77 shows the signal connection between the I2S0 module and the DAC. The DAC's control module regards I2S_CLK as the clock in this configuration. As shown in Figure 78, when the data bus inputs data to the DAC's control module, the latter will input right-channel data to DAC1 module and left-channel data to DAC2 module. When using the I2S DMA module, 8 bits of data-to-be-transmitted are shifted to the left by 8 bits of data-to-be-received into the DMA double-byte type of buffer.

The I2S_LCD_EN bit of register I2S_CONF2_REG should be set to 1, while I2S_RX_SHORT_SYNC, I2S_TX_SHORT_SYNC, I2S_CONF_REG , I2S_RX_MSB_SHIFT and I2S_TX_MSB_SHIFT should all be reset to 0. The I2S_TX_SLAVE_MOD bit of register I2S_CONF_REG should be set to 0, as well, when using the DAC mode of I2S0. Select a suitable transmit mode according to the standards of transmitting a 16-bit digital data stream. Configure the I2S0 module clock to output a suitable frequency for the I2S_CLK and the WS of I2S. Enable I2S0 to send data after configuring the relevant DAC registers.

12.6 I2S Interrupts

12.6.1 FIFO Interrupts

- I2S_TX_HUNG_INT: Triggered when transmitting data is timed out.
- I2S_RX_HUNG_INT: Triggered when receiving data is timed out.
- I2S_TX_REMPTY_INT: Triggered when the transmit FIFO is empty.
- I2S_TX_WFULL_INT: Triggered when the transmit FIFO is full.
- I2S_RX_REMPTY_INT: Triggered when the receive FIFO is empty.
- I2S_RX_WFULL_INT: Triggered when the receive FIFO is full.
- I2S_TX_PUT_DATA_INT: Triggered when the transmit FIFO is almost empty.
- I2S_RX_TAKE_DATA_INT: Triggered when the receive FIFO is almost full.

12.6.2 DMA Interrupts

- I2S_OUT_TOTAL_EOF_INT: Triggered when all transmitting linked lists are used up.
- I2S_IN_DSCR_EMPTY_INT: Triggered when there are no valid receiving linked lists left.
- I2S_OUT_DSCR_ERR_INT: Triggered when invalid rxlink descriptors are encountered.
- I2S_IN_DSCR_ERR_INT: Triggered when invalid txlink descriptors are encountered.
- I2S_OUT_EOF_INT: Triggered when rxlink has finished sending a packet.
- I2S_OUT_DONE_INT: Triggered when all transmitted and buffered data have been read.
- I2S_IN_SUC_EOF_INT: Triggered when all data have been received.
- I2S_IN_DONE_INT: Triggered when the current txlink descriptor is handled.

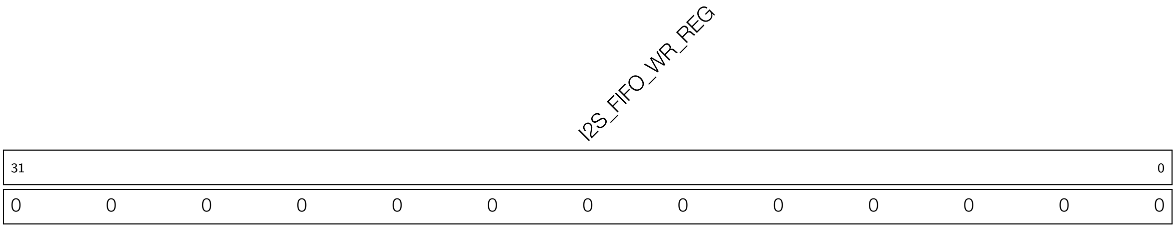
12.7 Register Summary

Name	Description	I2S0	I2S1	Acc
I2S FIFO registers				
I2S_FIFO_WR_REG	Writes the data sent by I2S into FIFO	0x3FF4F000	0x3FF6D000	WO
I2S_FIFO_RD_REG	Stores the data that I2S receives from FIFO	0x3FF4F004	0x3FF6D004	RO
Configuration registers				
I2S_CONF_REG	Configuration and start/stop bits	0x3FF4F008	0x3FF6D008	R/W
I2S_CONF1_REG	PCM configuration register	0x3FF4F0A0	0x3FF6D0A0	R/W
I2S_CONF2_REG	ADC/LCD/camera configuration register	0x3FF4F0A8	0x3FF6D0A8	R/W
I2S_TIMING_REG	Signal delay and timing parameters	0x3FF4F01C	0x3FF6D01C	R/W
I2S_FIFO_CONF_REG	FIFO configuration	0x3FF4F020	0x3FF6D020	R/W
I2S_CONF_SINGLE_DATA_REG	Static channel output value	0x3FF4F028	0x3FF6D028	R/W
I2S_CONF_CHAN_REG	Channel configuration	0x3FF4F02C	0x3FF6D02C	R/W
I2S_LC_HUNG_CONF_REG	Timeout detection configuration	0x3FF4F074	0x3FF6D074	R/W
I2S_CLKM_CONF_REG	Bitclock configuration	0x3FF4F0AC	0x3FF6D0AC	R/W
I2S_SAMPLE_RATE_CONF_REG	Sample rate configuration	0x3FF4F0B0	0x3FF6D0B0	R/W
I2S_PD_CONF_REG	Power-down register	0x3FF4F0A4	0x3FF6D0A4	R/W
I2S_STATE_REG	I2S status register	0x3FF4F0BC	0x3FF6D0BC	RO
DMA registers				
I2S_LC_CONF_REG	DMA configuration register	0x3FF4F060	0x3FF6D060	R/W
I2S_RXEOF_NUM_REG	Receive data count	0x3FF4F024	0x3FF6D024	R/W
I2S_OUT_LINK_REG	DMA transmit linked list configuration and address	0x3FF4F030	0x3FF6D030	R/W
I2S_IN_LINK_REG	DMA receive linked list configuration and address	0x3FF4F034	0x3FF6D034	R/W
I2S_OUT_EOF_DES_ADDR_REG	The address of transmit link descriptor producing EOF	0x3FF4F038	0x3FF6D038	RO
I2S_IN_EOF_DES_ADDR_REG	The address of receive link descriptor producing EOF	0x3FF4F03C	0x3FF6D03C	RO
I2S_OUT_EOF_BFR_DES_ADDR_REG	The address of transmit buffer producing EOF	0x3FF4F040	0x3FF6D040	RO
I2S_INLINK_DSCR_REG	The address of current inlink descriptor	0x3FF4F048	0x3FF6D048	RO
I2S_INLINK_DSCR_BF0_REG	The address of next inlink descriptor	0x3FF4F04C	0x3FF6D04C	RO
I2S_INLINK_DSCR_BF1_REG	The address of next inlink data buffer	0x3FF4F050	0x3FF6D050	RO
I2S_OUTLINK_DSCR_REG	The address of current outlink descriptor	0x3FF4F054	0x3FF6D054	RO
I2S_OUTLINK_DSCR_BF0_REG	The address of next outlink descriptor	0x3FF4F058	0x3FF6D058	RO

I2S_OUTLINK_DSCR_BF1_REG	The address of next outlink data buffer	0x3FF4F05C	0x3FF6D05C	RO
I2S_LC_STATE0_REG	DMA receive status	0x3FF4F06C	0x3FF6D06C	RO
I2S_LC_STATE1_REG	DMA transmit status	0x3FF4F070	0x3FF6D070	RO
Pulse density (DE) modulation registers				
I2S_PDM_CONF_REG	PDM configuration	0x3FF4F0B4	0x3FF6D0B4	R/W
I2S_PDM_FREQ_CONF_REG	PDM frequencies	0x3FF4F0B8	0x3FF6D0B8	R/W
Interrupt registers				
I2S_INT_RAW_REG	Raw interrupt status	0x3FF4F00C	0x3FF6D00C	RO
I2S_INT_ST_REG	Masked interrupt status	0x3FF4F010	0x3FF6D010	RO
I2S_INT_ENA_REG	Interrupt enable bits	0x3FF4F014	0x3FF6D014	R/W
I2S_INT_CLR_REG	Interrupt clear bits	0x3FF4F018	0x3FF6D018	WO

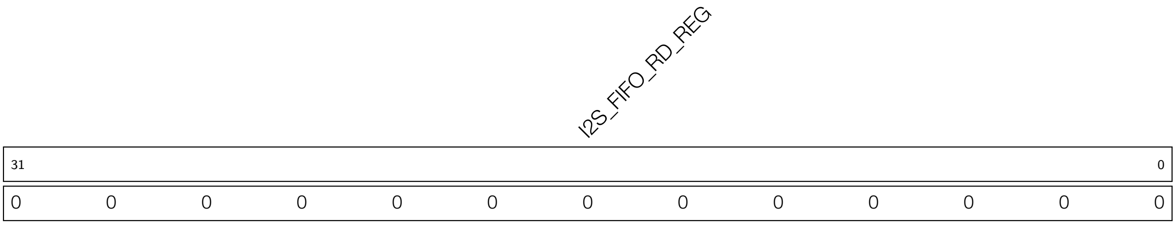
12.8 Registers

Register 12.1: I2S_FIFO_WR_REG (0x0000)



I2S_FIFO_WR_REG Writes the data sent by I2S into FIFO. (WO)

Register 12.2: I2S_FIFO_RD_REG (0x0004)



I2S_FIFO_RD_REG Stores the data that I2S receives from FIFO. (RO)

Register 12.3: I2S_CONF_REG (0x0008)

(reserved)	I2S_SIG_LOOPBACK	I2S_RX_MSB_RIGHT	I2S_TX_MSB_RIGHT	I2S_RX_MONO	I2S_TX_MONO	I2S_RX_SHORT_SYNC	I2S_TX_SHORT_SYNC	I2S_RX_MSB_SHIFT	I2S_TX_MSB_SHIFT	I2S_RX_RIGHT_FIRST	I2S_TX_RIGHT_FIRST	I2S_RX_SLAVE_MOD	I2S_TX_SLAVE_MOD	I2S_RX_START	I2S_TX_START	I2S_RX_FIFO_RESET	I2S_TX_FIFO_RESET	I2S_RX_RESET	I2S_TX_RESET	
31 19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_SIG_LOOPBACK Enable signal loopback mode, with transmitter module and receiver module sharing the same WS and BCK signals. (R/W)

I2S_RX_MSB_RIGHT Set this to place right-channel data at the MSB in the receive FIFO. (R/W)

I2S_TX_MSB_RIGHT Set this bit to place right-channel data at the MSB in the transmit FIFO. (R/W)

I2S_RX_MONO Set this bit to enable receiver's mono mode in PCM standard mode. (R/W)

I2S_TX_MONO Set this bit to enable transmitter's mono mode in PCM standard mode. (R/W)

I2S_RX_SHORT_SYNC Set this bit to enable receiver in PCM standard mode. (R/W)

I2S_TX_SHORT_SYNC Set this bit to enable transmitter in PCM standard mode. (R/W)

I2S_RX_MSB_SHIFT Set this bit to enable receiver in Philips standard mode. (R/W)

I2S_TX_MSB_SHIFT Set this bit to enable transmitter in Philips standard mode. (R/W)

I2S_RX_RIGHT_FIRST Set this bit to receive right-channel data first. (R/W)

I2S_TX_RIGHT_FIRST Set this bit to transmit right-channel data first. (R/W)

I2S_RX_SLAVE_MOD Set this bit to enable slave receiver mode. (R/W)

I2S_TX_SLAVE_MOD Set this bit to enable slave transmitter mode. (R/W)

I2S_RX_START Set this bit to start receiving data. (R/W)

I2S_TX_START Set this bit to start transmitting data. (R/W)

I2S_RX_FIFO_RESET Set this bit to reset the receive FIFO. (R/W)

I2S_TX_FIFO_RESET Set this bit to reset the transmit FIFO. (R/W)

I2S_RX_RESET Set this bit to reset the receiver. (R/W)

I2S_TX_RESET Set this bit to reset the transmitter. (R/W)

Register 12.4: I2S_INT_RAW_REG (0x000c)

(reserved)	I2S_OUT_TOTAL_EOF_INT_RAW	I2S_IN_DSCR_EMPTY_INT_RAW	I2S_OUT_DSCR_ERR_INT_RAW	I2S_IN_DSCR_ERR_INT_RAW	I2S_OUT_EOF_INT_RAW	I2S_OUT_DONE_INT_RAW	(reserved)	I2S_IN_SUC_EOF_INT_RAW	I2S_IN_DONE_INT_RAW	I2S_TX_HUNG_INT_RAW	I2S_RX_HUNG_INT_RAW	I2S_TX_REMPTY_INT_RAW	I2S_TX_WFULL_INT_RAW	I2S_RX_REMPTY_INT_RAW	I2S_RX_WFULL_INT_RAW	I2S_TX_PUT_DATA_INT_RAW	I2S_RX_TAKE_DATA_INT_RAW	
31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUT_TOTAL_EOF_INT_RAW The raw interrupt status bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

I2S_IN_DSCR_EMPTY_INT_RAW The raw interrupt status bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

I2S_OUT_DSCR_ERR_INT_RAW The raw interrupt status bit for the I2S_OUT_DSCR_ERR_INT interrupt. (RO)

I2S_IN_DSCR_ERR_INT_RAW The raw interrupt status bit for the I2S_IN_DSCR_ERR_INT interrupt. (RO)

I2S_OUT_EOF_INT_RAW The raw interrupt status bit for the I2S_OUT_EOF_INT interrupt. (RO)

I2S_OUT_DONE_INT_RAW The raw interrupt status bit for the I2S_OUT_DONE_INT interrupt. (RO)

I2S_IN_SUC_EOF_INT_RAW The raw interrupt status bit for the I2S_IN_SUC_EOF_INT interrupt. (RO)

I2S_IN_DONE_INT_RAW The raw interrupt status bit for the I2S_IN_DONE_INT interrupt. (RO)

I2S_TX_HUNG_INT_RAW The raw interrupt status bit for the I2S_TX_HUNG_INT interrupt. (RO)

I2S_RX_HUNG_INT_RAW The raw interrupt status bit for the I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_REMPTY_INT_RAW The raw interrupt status bit for the I2S_TX_REMPTY_INT interrupt. (RO)

I2S_TX_WFULL_INT_RAW The raw interrupt status bit for the I2S_TX_WFULL_INT interrupt. (RO)

I2S_RX_REMPTY_INT_RAW The raw interrupt status bit for the I2S_RX_REMPTY_INT interrupt. (RO)

I2S_RX_WFULL_INT_RAW The raw interrupt status bit for the I2S_RX_WFULL_INT interrupt. (RO)

I2S_TX_PUT_DATA_INT_RAW The raw interrupt status bit for the I2S_TX_PUT_DATA_INT interrupt. (RO)

I2S_RX_TAKE_DATA_INT_RAW The raw interrupt status bit for the I2S_RX_TAKE_DATA_INT interrupt. (RO)

Register 12.5: I2S_INT_ST_REG (0x0010)

	(reserved)	I2S_OUT_TOTAL_EOF_INT_ST	I2S_IN_DSCR_EMPTY_INT_ST	I2S_OUT_DSCR_ERR_INT_ST	I2S_IN_DSCR_ERR_INT_ST	I2S_OUT_EOF_INT_ST	I2S_OUT_DONE_INT_ST	(reserved)	I2S_IN_SUC_EOF_INT_ST	I2S_TX_DONE_INT_ST	I2S_TX_HUNG_INT_ST	I2S_RX_HUNG_INT_ST	I2S_TX_REMPTY_INT_ST	I2S_TX_WFULL_INT_ST	I2S_RX_REMPTY_INT_ST	I2S_RX_WFULL_INT_ST	I2S_TX_PUT_DATA_INT_ST	I2S_RX_TAKE_DATA_INT_ST
	31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2S_OUT_TOTAL_EOF_INT_ST The masked interrupt status bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (RO)

I2S_IN_DSCR_EMPTY_INT_ST The masked interrupt status bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (RO)

I2S_OUT_DSCR_ERR_INT_ST The masked interrupt status bit for the I2S_OUT_DSCR_ERR_INT interrupt. (RO)

I2S_IN_DSCR_ERR_INT_ST The masked interrupt status bit for the I2S_IN_DSCR_ERR_INT interrupt. (RO)

I2S_OUT_EOF_INT_ST The masked interrupt status bit for the I2S_OUT_EOF_INT interrupt. (RO)

I2S_OUT_DONE_INT_ST The masked interrupt status bit for the I2S_OUT_DONE_INT interrupt. (RO)

I2S_IN_SUC_EOF_INT_ST The masked interrupt status bit for the I2S_IN_SUC_EOF_INT interrupt. (RO)

I2S_IN_DONE_INT_ST The masked interrupt status bit for the I2S_IN_DONE_INT interrupt. (RO)

I2S_TX_HUNG_INT_ST The masked interrupt status bit for the I2S_TX_HUNG_INT interrupt. (RO)

I2S_RX_HUNG_INT_ST The masked interrupt status bit for the I2S_RX_HUNG_INT interrupt. (RO)

I2S_TX_REMPTY_INT_ST The masked interrupt status bit for the I2S_TX_REMPTY_INT interrupt. (RO)

I2S_TX_WFULL_INT_ST The masked interrupt status bit for the I2S_TX_WFULL_INT interrupt. (RO)

I2S_RX_REMPTY_INT_ST The masked interrupt status bit for the I2S_RX_REMPTY_INT interrupt. (RO)

I2S_RX_WFULL_INT_ST The masked interrupt status bit for the I2S_RX_WFULL_INT interrupt. (RO)

I2S_TX_PUT_DATA_INT_ST The masked interrupt status bit for the I2S_TX_PUT_DATA_INT interrupt. (RO)

I2S_RX_TAKE_DATA_INT_ST The masked interrupt status bit for the I2S_RX_TAKE_DATA_INT interrupt. (RO)

Register 12.6: I2S_INT_ENA_REG (0x0014)

(reserved)	I2S_OUT_TOTAL_EOF_INT_ENA	I2S_IN_DSCR_EMPTY_INT_ENA	I2S_OUT_DSCR_ERR_INT_ENA	I2S_IN_DSCR_ERR_INT_ENA	I2S_OUT_EOF_INT_ENA	I2S_OUT_DONE_INT_ENA	(reserved)	I2S_IN_SUC_EOF_INT_ENA	I2S_IN_DONE_INT_ENA	I2S_TX_HUNG_INT_ENA	I2S_RX_HUNG_INT_ENA	I2S_TX_REMPTY_INT_ENA	I2S_TX_WFULL_INT_ENA	I2S_RX_REMPTY_INT_ENA	I2S_RX_WFULL_INT_ENA	I2S_TX_PUT_DATA_INT_ENA	I2S_RX_TAKE_DATA_INT_ENA	
31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUT_TOTAL_EOF_INT_ENA The interrupt enable bit for the I2S_OUT_TOTAL_EOF_INT interrupt. (R/W)

I2S_IN_DSCR_EMPTY_INT_ENA The interrupt enable bit for the I2S_IN_DSCR_EMPTY_INT interrupt. (R/W)

I2S_OUT_DSCR_ERR_INT_ENA The interrupt enable bit for the I2S_OUT_DSCR_ERR_INT interrupt. (R/W)

I2S_IN_DSCR_ERR_INT_ENA The interrupt enable bit for the I2S_IN_DSCR_ERR_INT interrupt. (R/W)

I2S_OUT_EOF_INT_ENA The interrupt enable bit for the I2S_OUT_EOF_INT interrupt. (R/W)

I2S_OUT_DONE_INT_ENA The interrupt enable bit for the I2S_OUT_DONE_INT interrupt. (R/W)

I2S_IN_SUC_EOF_INT_ENA The interrupt enable bit for the I2S_IN_SUC_EOF_INT interrupt. (R/W)

I2S_IN_DONE_INT_ENA The interrupt enable bit for the I2S_IN_DONE_INT interrupt. (R/W)

I2S_TX_HUNG_INT_ENA The interrupt enable bit for the I2S_TX_HUNG_INT interrupt. (R/W)

I2S_RX_HUNG_INT_ENA The interrupt enable bit for the I2S_RX_HUNG_INT interrupt. (R/W)

I2S_TX_REMPTY_INT_ENA The interrupt enable bit for the I2S_TX_REMPTY_INT interrupt. (R/W)

I2S_TX_WFULL_INT_ENA The interrupt enable bit for the I2S_TX_WFULL_INT interrupt. (R/W)

I2S_RX_REMPTY_INT_ENA The interrupt enable bit for the I2S_RX_REMPTY_INT interrupt. (R/W)

I2S_RX_WFULL_INT_ENA The interrupt enable bit for the I2S_RX_WFULL_INT interrupt. (R/W)

I2S_TX_PUT_DATA_INT_ENA The interrupt enable bit for the I2S_TX_PUT_DATA_INT interrupt. (R/W)

I2S_RX_TAKE_DATA_INT_ENA The interrupt enable bit for the I2S_RX_TAKE_DATA_INT interrupt. (R/W)

Register 12.7: I2S_INT_CLR_REG (0x0018)

(reserved)	I2S_OUT_TOTAL_EOF_INT_CLR	I2S_IN_DSCR_EMPTY_INT_CLR	I2S_OUT_DSCR_ERR_INT_CLR	I2S_IN_DSCR_ERR_INT_CLR	I2S_OUT_EOF_INT_CLR	I2S_OUT_DONE_INT_CLR	(reserved)	I2S_IN_SUC_EOF_INT_CLR	I2S_IN_DONE_INT_CLR	I2S_TX_HUNG_INT_CLR	I2S_RX_HUNG_INT_CLR	I2S_TX_REMPTY_INT_CLR	I2S_TX_WFULL_INT_CLR	I2S_RX_REMPTY_INT_CLR	I2S_RX_WFULL_INT_CLR	I2S_TX_PUT_DATA_INT_CLR	I2S_RX_TAKE_DATA_INT_CLR	
31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUT_TOTAL_EOF_INT_CLR Set this bit to clear the I2S_OUT_TOTAL_EOF_INT interrupt. (WO)

I2S_IN_DSCR_EMPTY_INT_CLR Set this bit to clear the I2S_IN_DSCR_EMPTY_INT interrupt. (WO)

I2S_OUT_DSCR_ERR_INT_CLR Set this bit to clear the I2S_OUT_DSCR_ERR_INT interrupt. (WO)

I2S_IN_DSCR_ERR_INT_CLR Set this bit to clear the I2S_IN_DSCR_ERR_INT interrupt. (WO)

I2S_OUT_EOF_INT_CLR Set this bit to clear the I2S_OUT_EOF_INT interrupt. (WO)

I2S_OUT_DONE_INT_CLR Set this bit to clear the I2S_OUT_DONE_INT interrupt. (WO)

I2S_IN_SUC_EOF_INT_CLR Set this bit to clear the I2S_IN_SUC_EOF_INT interrupt. (WO)

I2S_IN_DONE_INT_CLR Set this bit to clear the I2S_IN_DONE_INT interrupt. (WO)

I2S_TX_HUNG_INT_CLR Set this bit to clear the I2S_TX_HUNG_INT interrupt. (WO)

I2S_RX_HUNG_INT_CLR Set this bit to clear the I2S_RX_HUNG_INT interrupt. (WO)

I2S_TX_REMPTY_INT_CLR Set this bit to clear the I2S_TX_REMPTY_INT interrupt. (WO)

I2S_TX_WFULL_INT_CLR Set this bit to clear the I2S_TX_WFULL_INT interrupt. (WO)

I2S_RX_REMPTY_INT_CLR Set this bit to clear the I2S_RX_REMPTY_INT interrupt. (WO)

I2S_RX_WFULL_INT_CLR Set this bit to clear the I2S_RX_WFULL_INT interrupt. (WO)

I2S_TX_PUT_DATA_INT_CLR Set this bit to clear the I2S_TX_PUT_DATA_INT interrupt. (WO)

I2S_RX_TAKE_DATA_INT_CLR Set this bit to clear the I2S_RX_TAKE_DATA_INT interrupt. (WO)

Register 12.8: I2S_TIMING_REG (0x001c)

(reserved)	I2S_TX_BCK_IN_INV	I2S_DATA_ENABLE_DELAY	I2S_RX_DSYNC_SW	I2S_TX_DSYNC_SW	I2S_RX_BCK_OUT_DELAY	I2S_RX_WS_OUT_DELAY	I2S_TX_SD_OUT_DELAY	I2S_TX_WS_OUT_DELAY	I2S_TX_BCK_OUT_DELAY	I2S_RX_SD_IN_DELAY	I2S_RX_WS_IN_DELAY	I2S_RX_BCK_IN_DELAY	I2S_TX_WS_IN_DELAY	I2S_TX_BCK_IN_DELAY	
31 25	24	23 22	21	20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0	0	0 0	0	0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	Reset

I2S_TX_BCK_IN_INV Set this bit to invert the BCK signal into the slave transmitter. (R/W)

I2S_DATA_ENABLE_DELAY Number of delay cycles for data valid flag. (R/W)

I2S_RX_DSYNC_SW Set this bit to synchronize signals into the receiver in double sync method. (R/W)

I2S_TX_DSYNC_SW Set this bit to synchronize signals into the transmitter in double sync method. (R/W)

I2S_RX_BCK_OUT_DELAY Number of delay cycles for BCK signal out of the receiver. (R/W)

I2S_RX_WS_OUT_DELAY Number of delay cycles for WS signal out of the receiver. (R/W)

I2S_TX_SD_OUT_DELAY Number of delay cycles for SD signal out of the transmitter. (R/W)

I2S_TX_WS_OUT_DELAY Number of delay cycles for WS signal out of the transmitter. (R/W)

I2S_TX_BCK_OUT_DELAY Number of delay cycles for BCK signal out of the transmitter. (R/W)

I2S_RX_SD_IN_DELAY Number of delay cycles for SD signal into the receiver. (R/W)

I2S_RX_WS_IN_DELAY Number of delay cycles for WS signal into the receiver. (R/W)

I2S_RX_BCK_IN_DELAY Number of delay cycles for BCK signal into the receiver. (R/W)

I2S_TX_WS_IN_DELAY Number of delay cycles for WS signal into the transmitter. (R/W)

I2S_TX_BCK_IN_DELAY Number of delay cycles for BCK signal into the transmitter. (R/W)

Register 12.9: I2S_FIFO_CONF_REG (0x0020)

(reserved)	I2S_RX_FIFO_MOD_FORCE_EN	I2S_TX_FIFO_MOD_FORCE_EN	I2S_RX_FIFO_MOD	I2S_TX_FIFO_MOD	I2S_DSCR_EN	I2S_TX_DATA_NUM	I2S_RX_DATA_NUM	
31 21	20	19	18 16	15 13	12	11 6	5 0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0	0 0 0	1	32	32	Reset

I2S_RX_FIFO_MOD_FORCE_EN The bit should always be set to 1. (R/W)

I2S_TX_FIFO_MOD_FORCE_EN The bit should always be set to 1. (R/W)

I2S_RX_FIFO_MOD Receive FIFO mode configuration bit. (R/W)

I2S_TX_FIFO_MOD Transmit FIFO mode configuration bit. (R/W)

I2S_DSCR_EN Set this bit to enable I2S DMA mode. (R/W)

I2S_TX_DATA_NUM Threshold of data length in the transmit FIFO. (R/W)

I2S_RX_DATA_NUM Threshold of data length in the receive FIFO. (R/W)

Register 12.10: I2S_RXEOF_NUM_REG (0x0024)

31 0	
64	Reset

I2S_RXEOF_NUM_REG The length of the data to be received. It will trigger I2S_IN_SUC_EOF_INT. (R/W)

Register 12.11: I2S_CONF_SINGLE_DATA_REG (0x0028)

31 0	
0	Reset

I2S_CONF_SINGLE_DATA_REG The right channel or the left channel outputs constant values stored in this register according to TX_CHAN_MOD and I2S_TX_MSB_RIGHT. (R/W)

Register 12.12: I2S_CONF_CHAN_REG (0x002c)

(reserved)	I2S_RX_CHAN_MOD	I2S_TX_CHAN_MOD	
31 5	4 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0	Reset

I2S_RX_CHAN_MOD I2S receiver channel mode configuration bits. Please refer to Section 12.4.5 for further details. (R/W)

I2S_TX_CHAN_MOD I2S transmitter channel mode configuration bits. Please refer to Section 12.4.4 for further details. (R/W)

Register 12.13: I2S_OUT_LINK_REG (0x0030)

(reserved)	I2S_OUTLINK_RESTART	I2S_OUTLINK_START	I2S_OUTLINK_STOP	(reserved)	I2S_OUTLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

I2S_OUTLINK_RESTART Set this bit to restart outlink descriptor. (R/W)

I2S_OUTLINK_START Set this bit to start outlink descriptor. (R/W)

I2S_OUTLINK_STOP Set this bit to stop outlink descriptor. (R/W)

I2S_OUTLINK_ADDR The address of first outlink descriptor. (R/W)

Register 12.14: I2S_IN_LINK_REG (0x0034)

(reserved)	I2S_INLINK_RESTART	I2S_INLINK_START	I2S_INLINK_STOP	(reserved)	I2S_INLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

I2S_INLINK_RESTART Set this bit to restart inlink descriptor. (R/W)

I2S_INLINK_START Set this bit to start inlink descriptor. (R/W)

I2S_INLINK_STOP Set this bit to stop inlink descriptor. (R/W)

I2S_INLINK_ADDR The address of first inlink descriptor. (R/W)

Register 12.15: I2S_OUT_EOF_DES_ADDR_REG (0x0038)

31	0	
0x000000000		Reset

I2S_OUT_EOF_DES_ADDR_REG The address of outlink descriptor that produces EOF. (RO)

Register 12.16: I2S_IN_EOF_DES_ADDR_REG (0x003c)

31	0	
0x000000000		Reset

I2S_IN_EOF_DES_ADDR_REG The address of inlink descriptor that produces EOF. (RO)

Register 12.17: I2S_OUT_EOF_BFR_DES_ADDR_REG (0x0040)

31	0	
0x000000000		Reset

I2S_OUT_EOF_BFR_DES_ADDR_REG The address of the buffer corresponding to the outlink descriptor that produces EOF. (RO)

Register 12.18: I2S_INLINK_DSCR_REG (0x0048)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_INLINK_DSCR_REG The address of current inlink descriptor. (RO)

Register 12.19: I2S_INLINK_DSCR_BF0_REG (0x004c)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_INLINK_DSCR_BF0_REG The address of next inlink descriptor. (RO)

Register 12.20: I2S_INLINK_DSCR_BF1_REG (0x0050)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_INLINK_DSCR_BF1_REG The address of next inlink data buffer. (RO)

Register 12.21: I2S_OUTLINK_DSCR_REG (0x0054)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_REG The address of current outlink descriptor. (RO)

Register 12.22: I2S_OUTLINK_DSCR_BF0_REG (0x0058)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_BF0_REG The address of next outlink descriptor. (RO)

Register 12.23: I2S_OUTLINK_DSCR_BF1_REG (0x005c)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_BF1_REG The address of next outlink data buffer. (RO)

Register 12.24: I2S_LC_CONF_REG (0x0060)

(reserved)	I2S_CHECK_OWNER	I2S_OUT_DATA_BURST_EN	I2S_INDSCR_BURST_EN	I2S_OUTDSCR_BURST_EN	I2S_OUT_EOF_MODE	(reserved)	I2S_OUT_AUTO_WRBACK	I2S_OUT_LOOP_TEST	I2S_IN_LOOP_TEST	I2S_AHBM_RST	I2S_AHBM_FIFO_RST	I2S_OUT_RST	I2S_IN_RST	
31 … 13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	1	0	0	0	0	0	0	0	0	Reset

I2S_CHECK_OWNER Set this bit to check the owner bit by hardware. (R/W)

I2S_OUT_DATA_BURST_EN Transmitter data transfer mode configuration bit. (R/W)

1: Transmit data in burst mode;

0: Transmit data in byte mode.

I2S_INDSCR_BURST_EN DMA inlink descriptor transfer mode configuration bit. (R/W)

1: Transfer inlink descriptor in burst mode;

0: Transfer inlink descriptor in byte mode.

I2S_OUTDSCR_BURST_EN DMA outlink descriptor transfer mode configuration bit. (R/W)

1: Transfer outlink descriptor in burst mode;

0: Transfer outlink descriptor in byte mode.

I2S_OUT_EOF_MODE DMA I2S_OUT_EOF_INT generation mode. (R/W)

1: When DMA has popped all data from the FIFO;

0: When AHB has pushed all data to the FIFO.

I2S_OUT_AUTO_WRBACK Set this bit to enable automatic outlink-writeback when all the data in tx buffer has been transmitted. (R/W)

I2S_OUT_LOOP_TEST Set this bit to loop test outlink. (R/W)

I2S_IN_LOOP_TEST Set this bit to loop test inlink. (R/W)

I2S_AHBM_RST Set this bit to reset AHB interface of DMA. (R/W)

I2S_AHBM_FIFO_RST Set this bit to reset AHB interface cmdFIFO of DMA. (R/W)

I2S_OUT_RST Set this bit to reset out DMA FSM. (R/W)

I2S_IN_RST Set this bit to reset in DMA FSM. (R/W)

Register 12.25: I2S_LC_STATE0_REG (0x006c)

31 … 0	
0x000000000	Reset

I2S_LC_STATE0_REG Receiver DMA channel status register. (RO)

Register 12.26: I2S_LC_STATE1_REG (0x0070)

31 0	
0x000000000	Reset

I2S_LC_STATE1_REG Transmitter DMA channel status register. (RO)

Register 12.27: I2S_LC_HUNG_CONF_REG (0x0074)

(reserved)	I2S_LC_FIFO_TIMEOUT_ENA	I2S_LC_FIFO_TIMEOUT_SHIFT	I2S_LC_FIFO_TIMEOUT	
31 12	11	10 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	0x010	Reset

I2S_LC_FIFO_TIMEOUT_ENA The enable bit for FIFO timeout. (R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT The bits are used to set the tick counter threshold. The tick counter is reset when the counter value >= $88000/2^{i2s\_lc\_fifo\_timeout\_shift}$. (R/W)

I2S_LC_FIFO_TIMEOUT When the value of FIFO hung counter is equal to this bit value, sending data-timeout interrupt or receiving data-timeout interrupt will be triggered. (R/W)